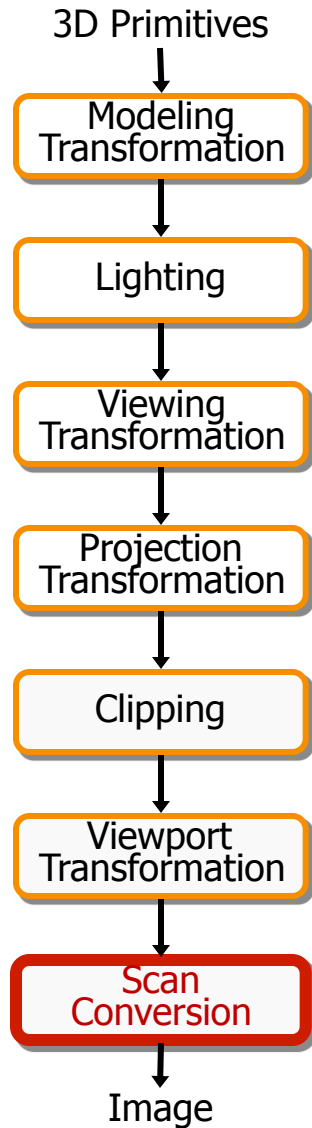




Rasterization

COS 426, Spring 2015
Princeton University

3D Rendering Pipeline (for direct illumination)





Rasterization

- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

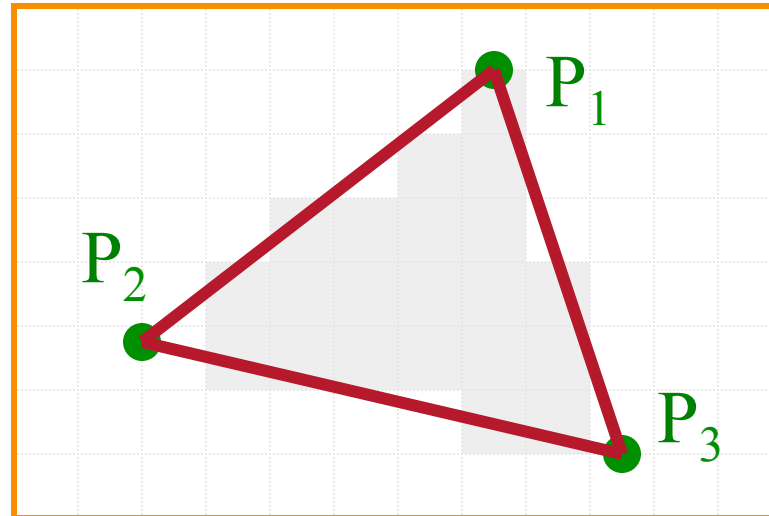


Rasterization

- Scan conversion (last time)
 - Determine which pixels to fill
- **Shading**
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Shading

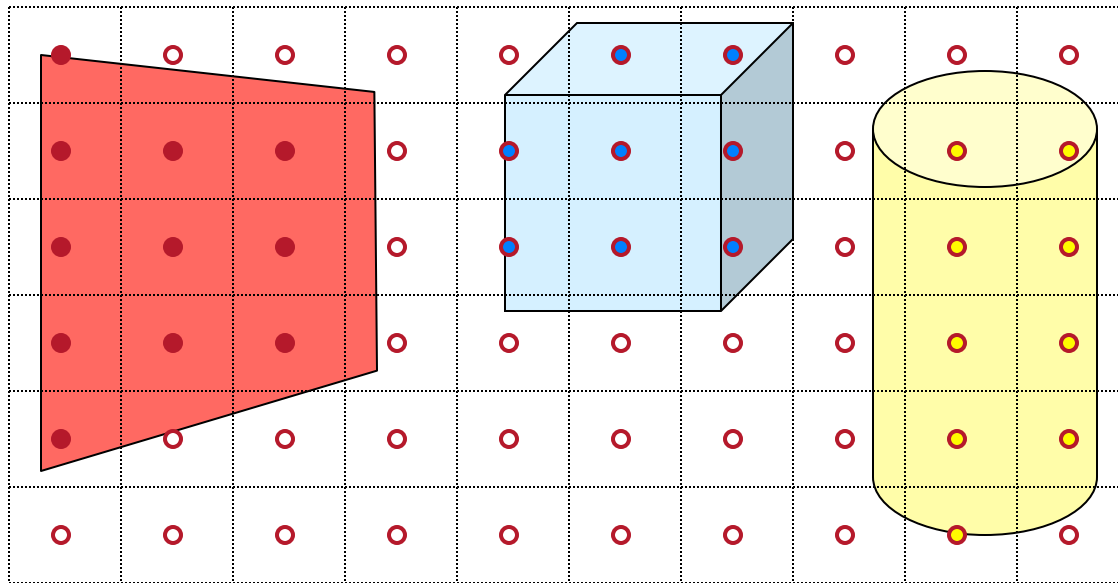
- How do we choose a color for each filled pixel?



Emphasis on methods that can be implemented in hardware

Ray Casting

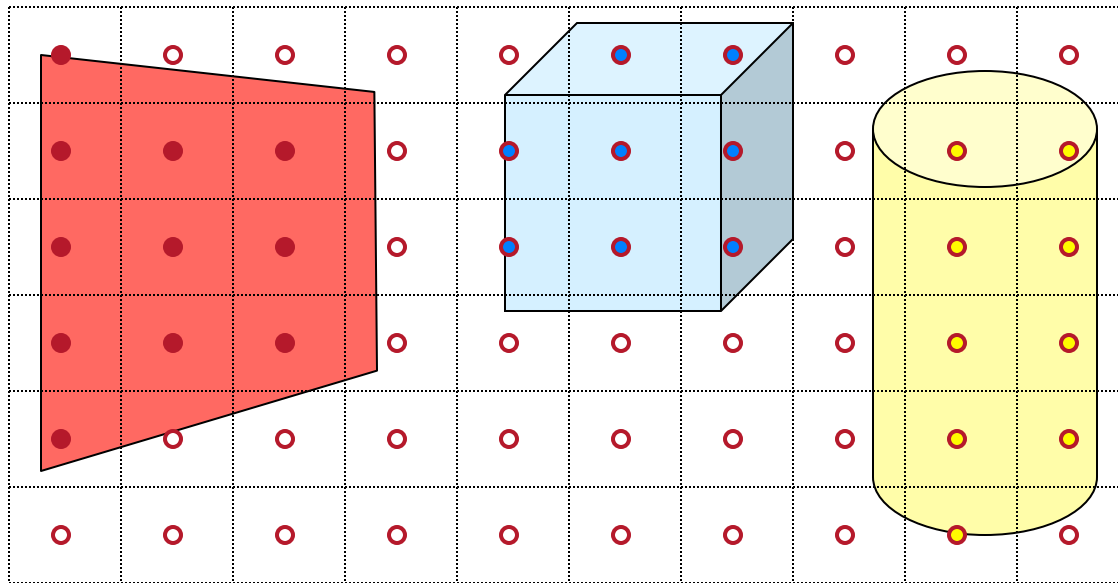
- Simplest shading approach is to perform independent lighting calculation for every pixel



$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$

Polygon Shading

- Can take advantage of spatial coherence
 - Illumination calculations for pixels covered by same primitive are related to each other



$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$

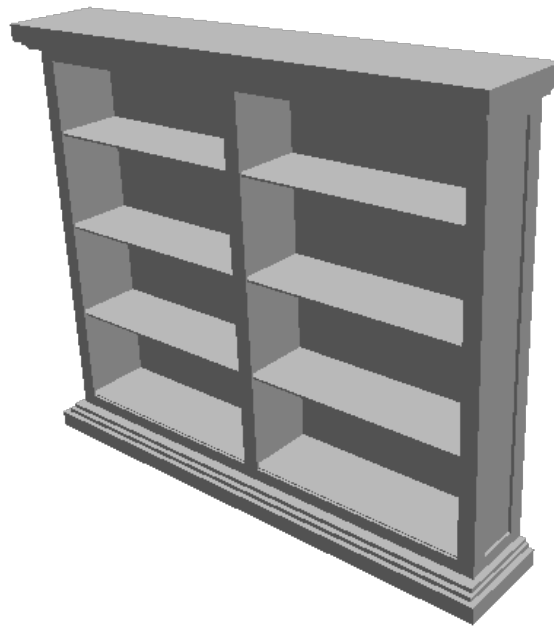
Polygon Shading Algorithms



- **Flat Shading**
- Gouraud Shading
- Phong Shading

Flat Shading

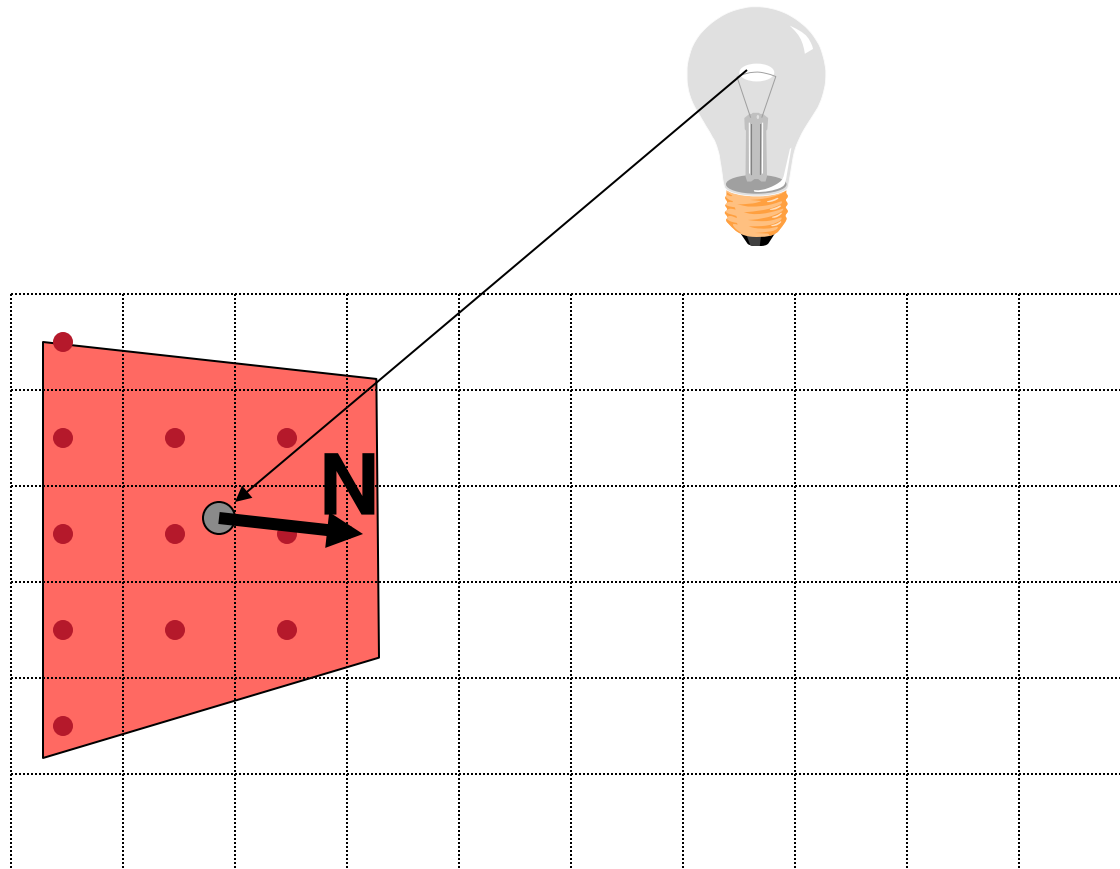
- What if a faceted object is illuminated only by directional light sources and is either diffuse or viewed from infinitely far away



$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$

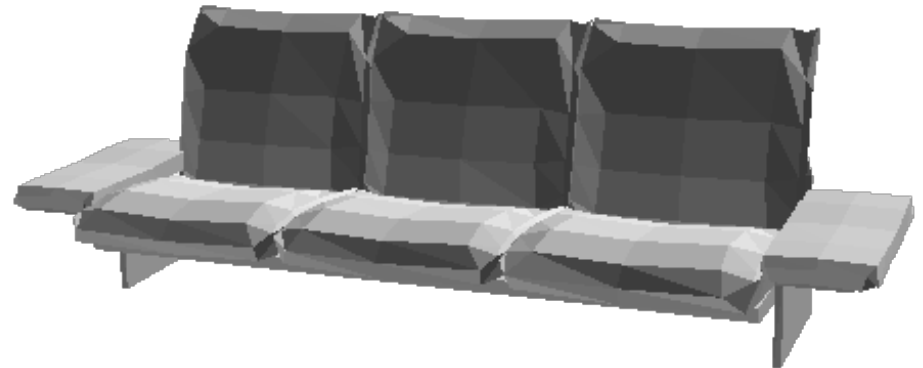
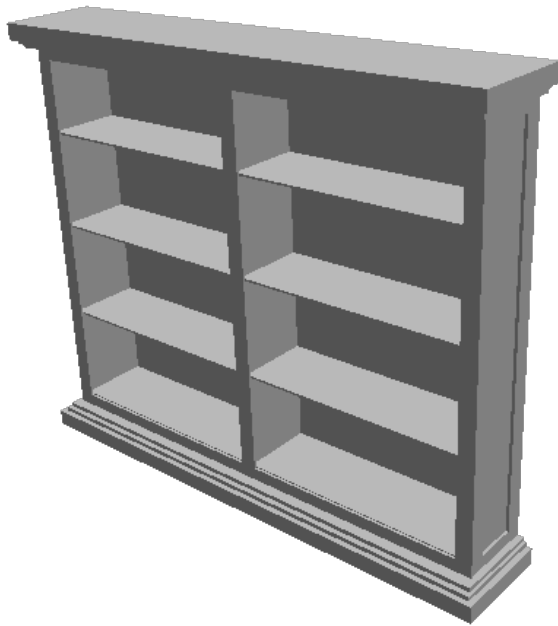
Flat Shading

- One illumination calculation per polygon
 - Assign all pixels inside each polygon the same color



Flat Shading

- Objects look like they are composed of polygons
 - OK for polyhedral objects
 - Not so good for smooth surfaces



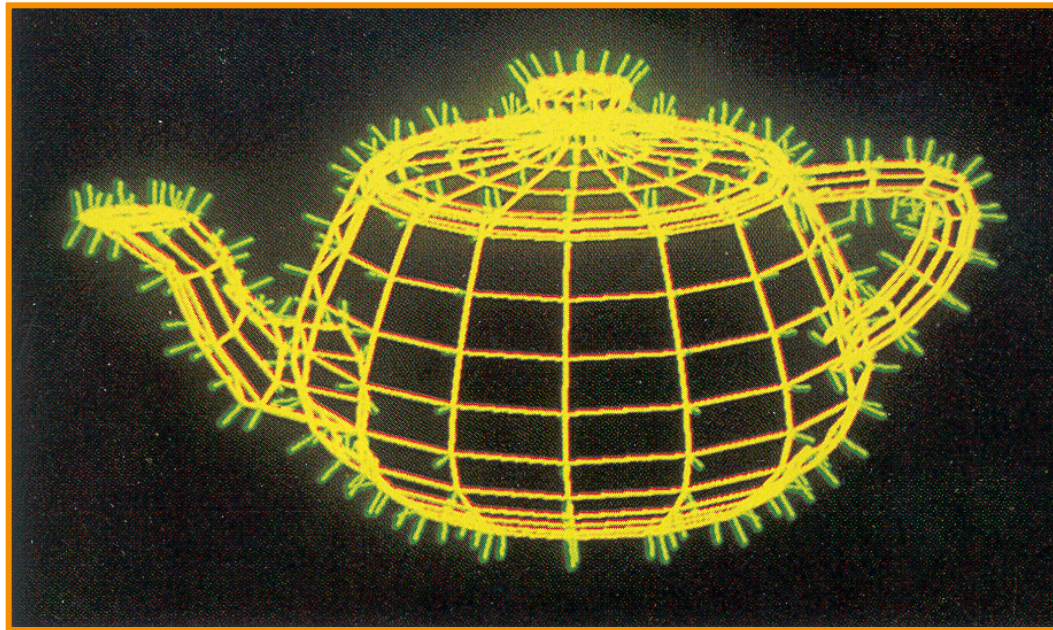
Polygon Shading Algorithms



- Flat Shading
- **Gouraud Shading**
- Phong Shading

Gouraud Shading

- What if smooth surface is represented by polygonal mesh with a normal at each vertex?

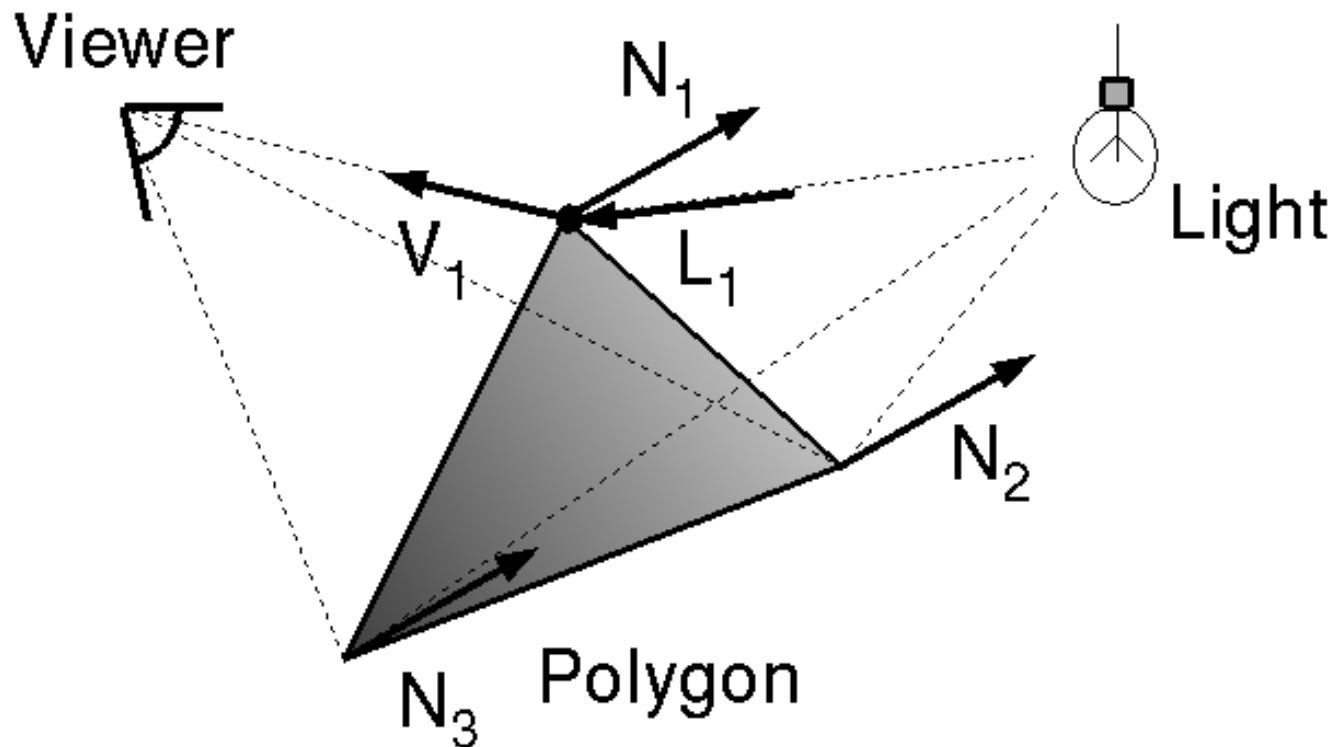


Watt Plate 7

$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$

Gouraud Shading

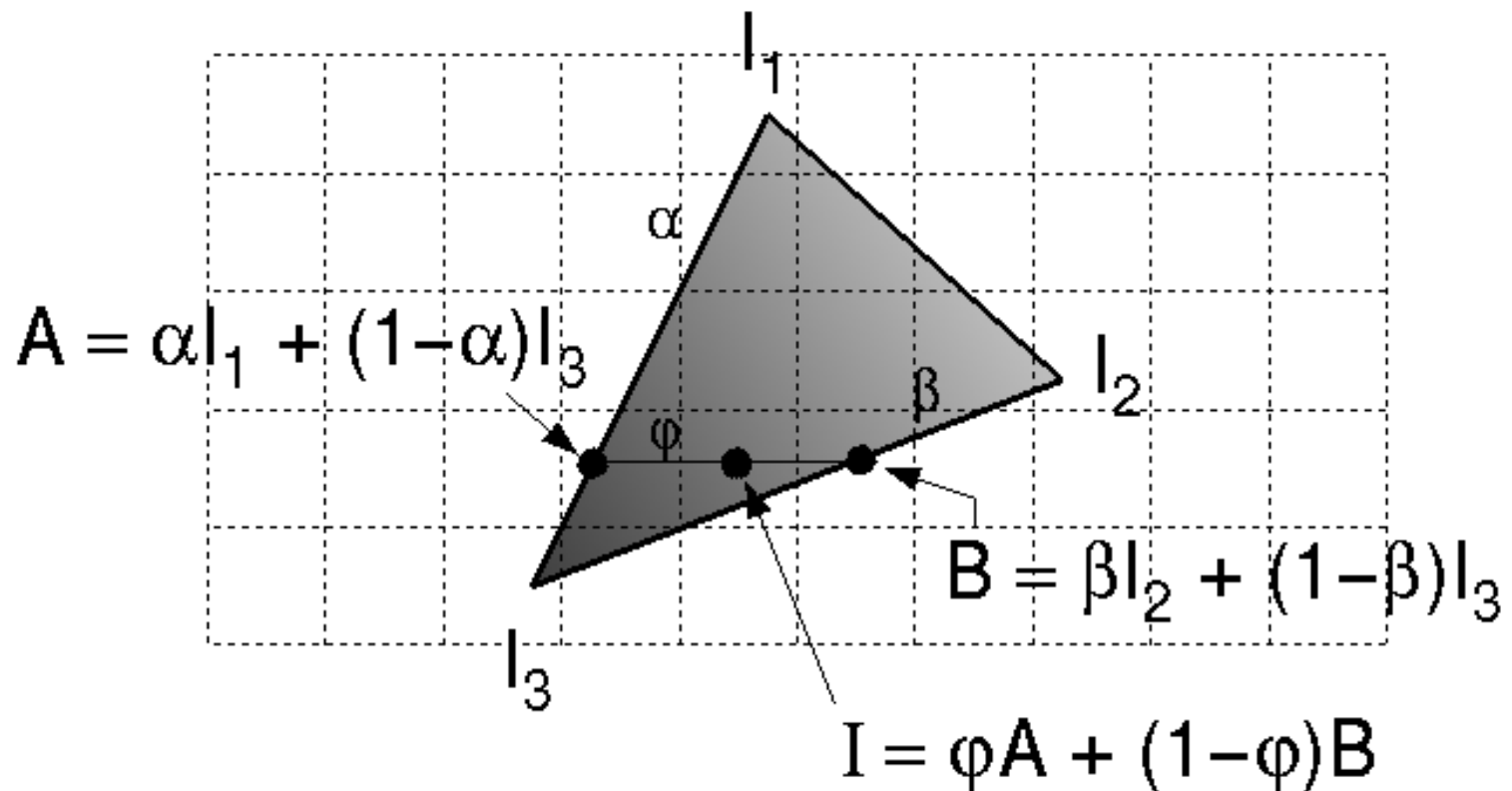
- Method 1: One lighting calculation per vertex
 - Assign pixels inside polygon by interpolating colors computed at vertices





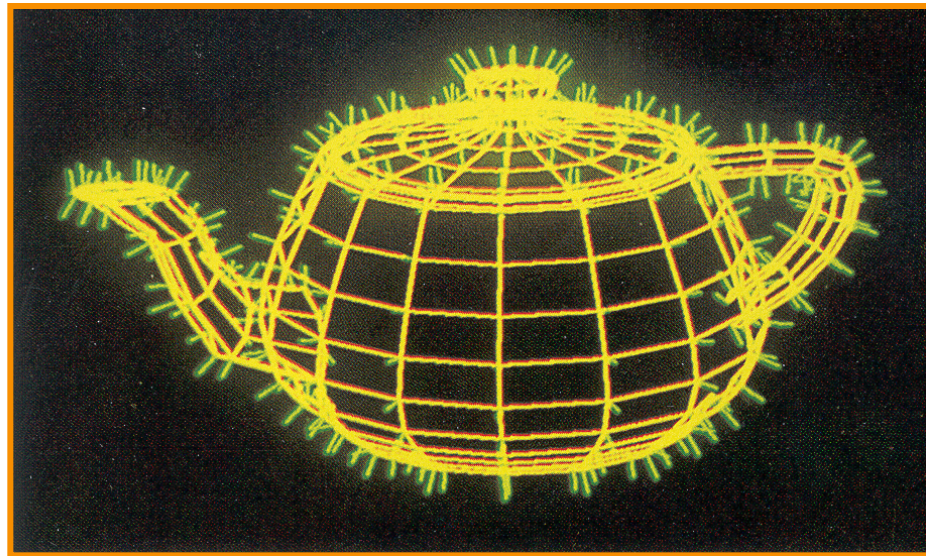
Gouraud Shading

- Bilinearly interpolate colors at vertices down and across scan lines



Gouraud Shading

- Smooth shading over adjacent polygons
 - Curved surfaces
 - Illumination highlights
 - Soft shadows

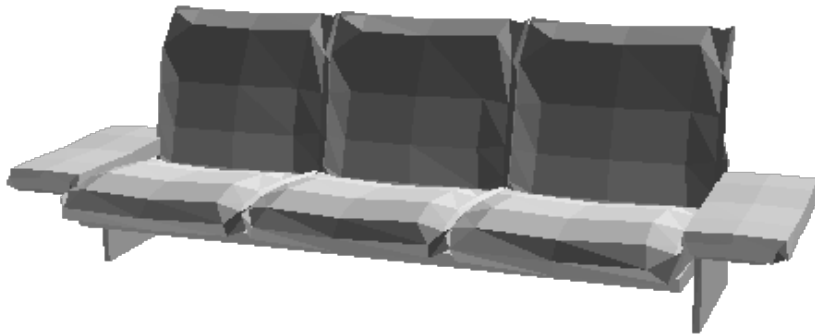


Mesh with shared normals at vertices

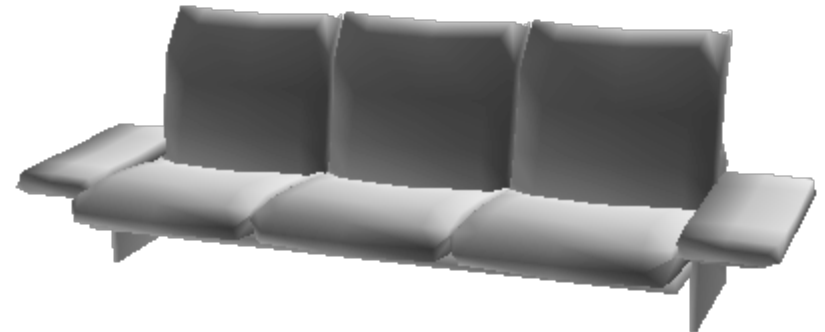
Gouraud Shading



- Produces smoothly shaded polygonal mesh
 - Piecewise linear approximation
 - Need fine mesh to capture subtle lighting effects



Flat Shading



Gouraud Shading

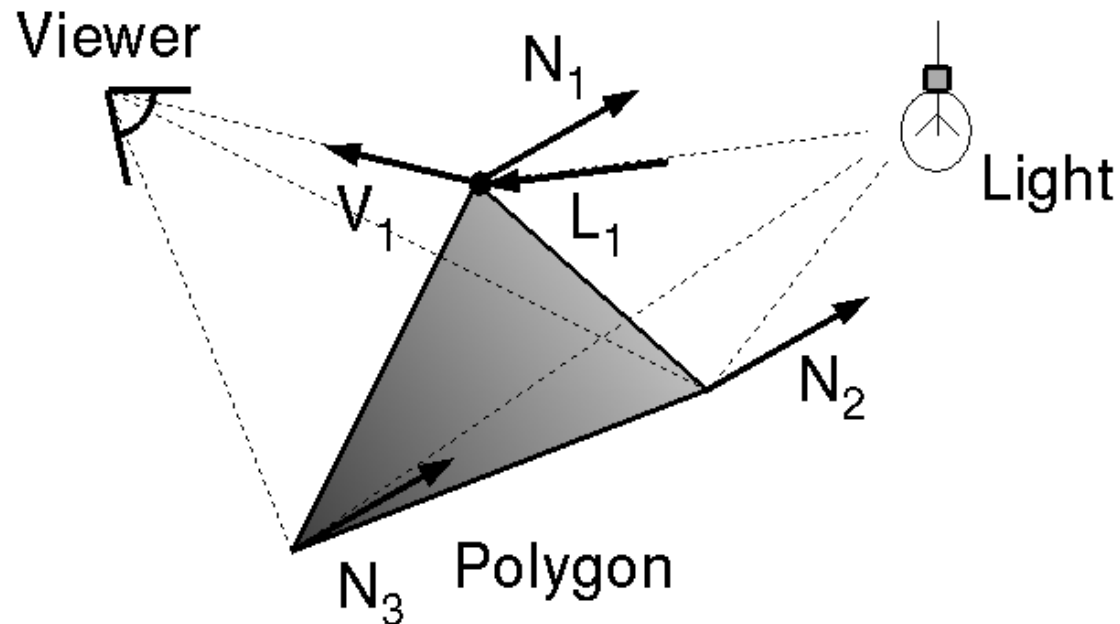
Polygon Shading Algorithms



- Flat Shading
- Gouraud Shading
- **Phong Shading** (\neq Phong reflectance model)

Phong Shading

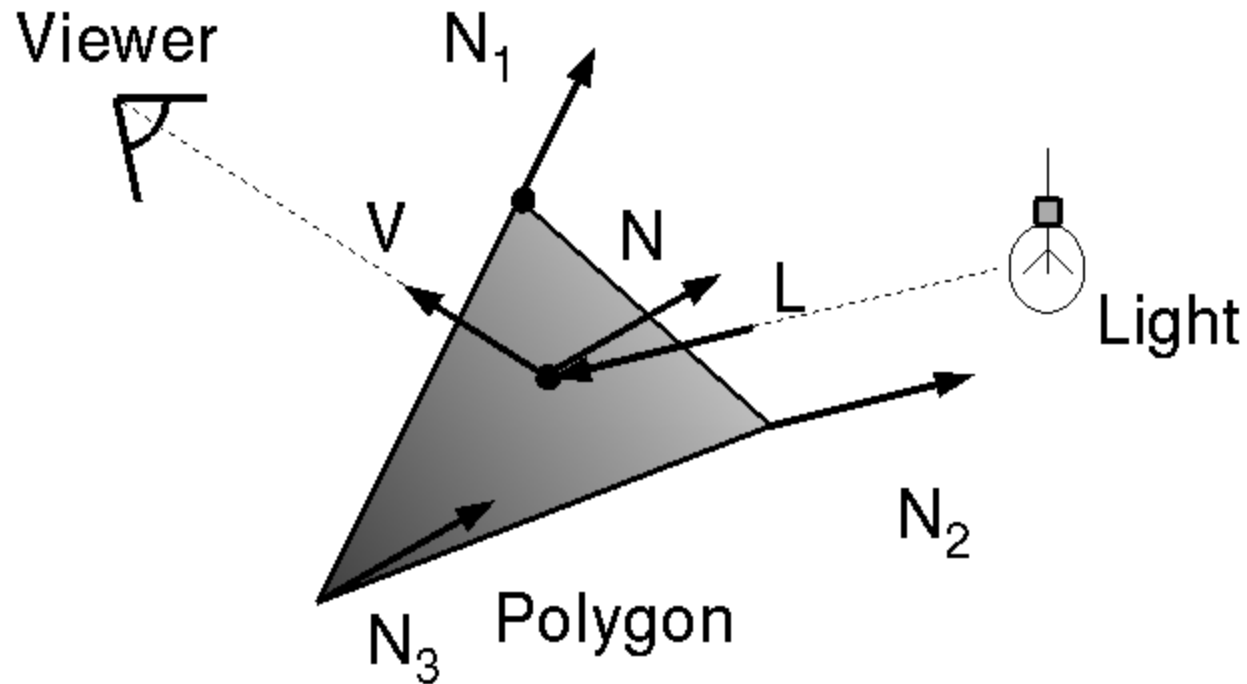
- What if polygonal mesh is too coarse to capture illumination effects in polygon interiors?



$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$

Phong Shading

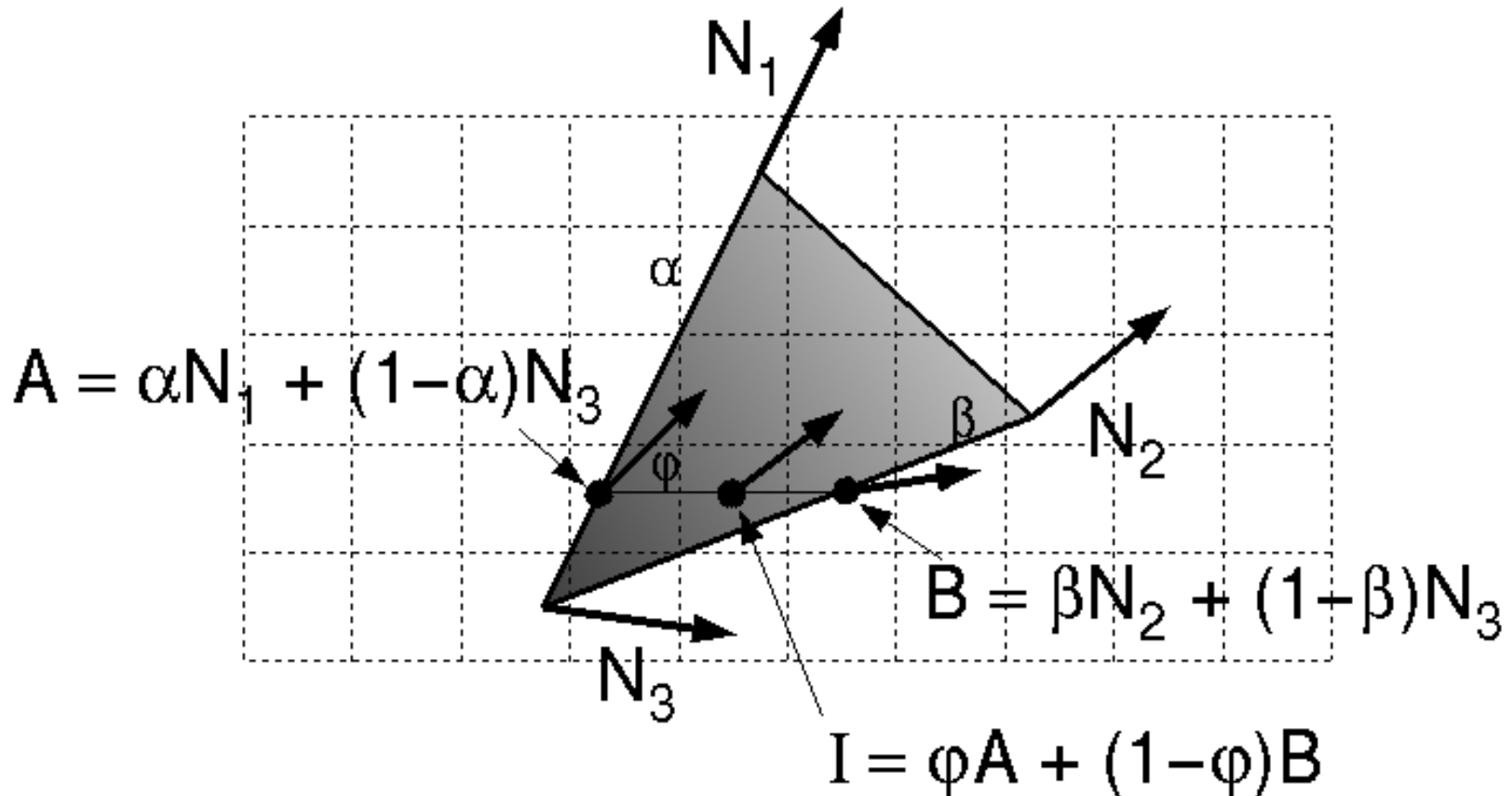
- One lighting calculation per pixel
 - Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices





Phong Shading

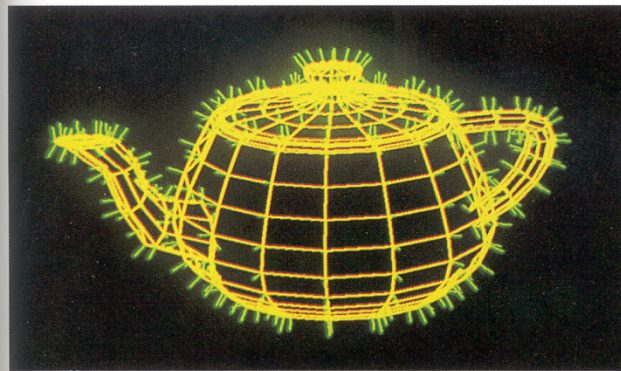
- Bilinearly interpolate surface normals at vertices down and across scan lines



Polygon Shading Algorithms



Wireframe



Flat



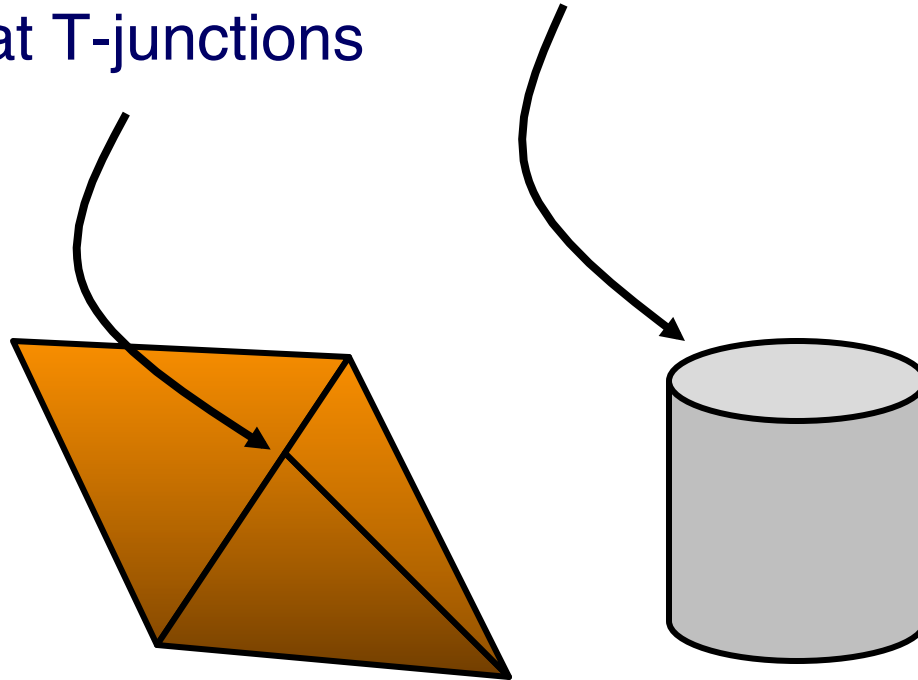
Gouraud



Phong

Shading Issues

- Problems with interpolated shading:
 - Polygonal silhouettes
 - Perspective distortion (due to screen-space interpolation)
 - Problems computing shared vertex normals
 - Problems at T-junctions





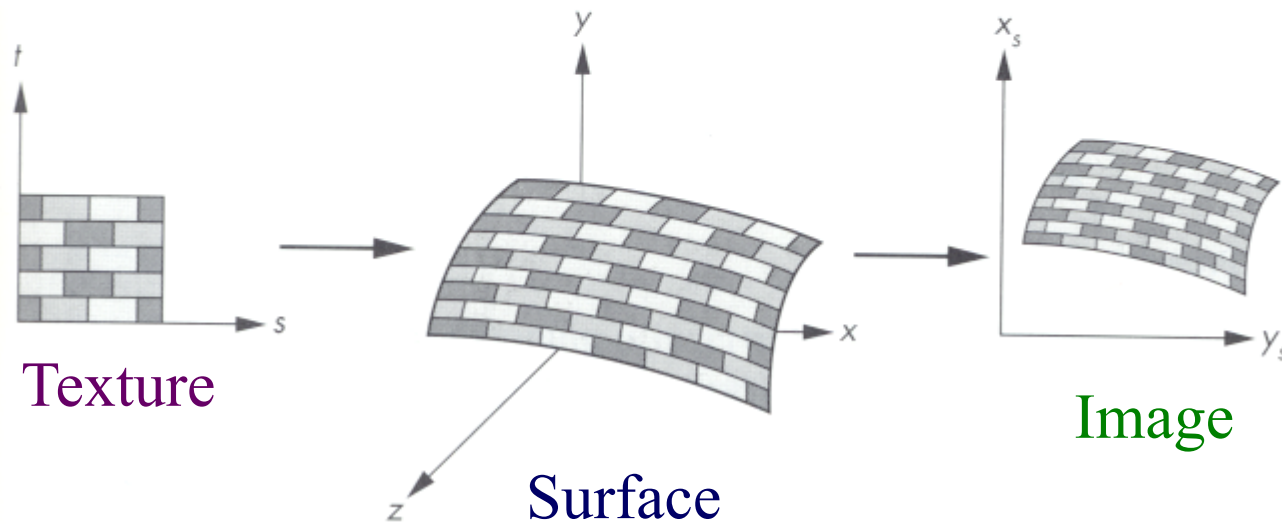
Rasterization

- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- **Texture mapping**
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Textures



- Describe color variation in interior of 3D polygon
 - When scan converting a polygon, vary pixel colors according to values fetched from a texture image



Surface Textures



- Add visual detail to surfaces of 3D objects



[Daren Horley]

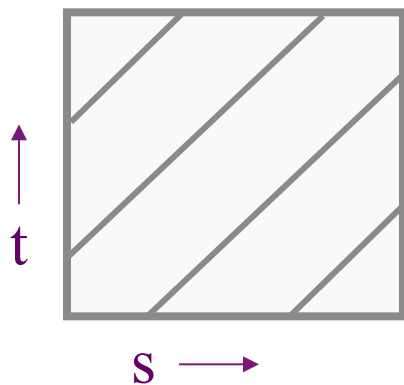


Texture Mapping Overview

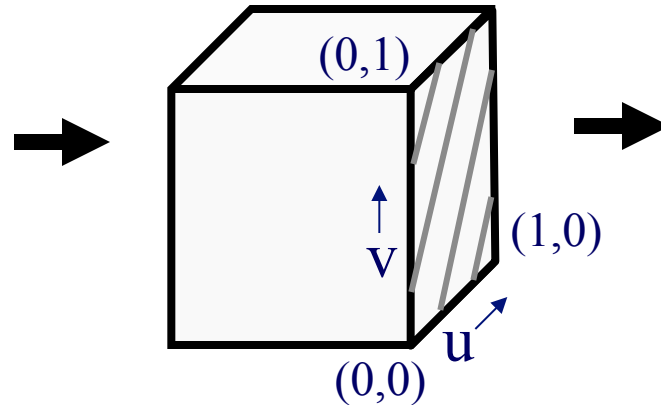
- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering

Texture Mapping

- Steps:
 - Define texture
 - Specify mapping from texture to surface
 - Look up texture values during scan conversion



Texture
Coordinate
System



Modeling
Coordinate
System

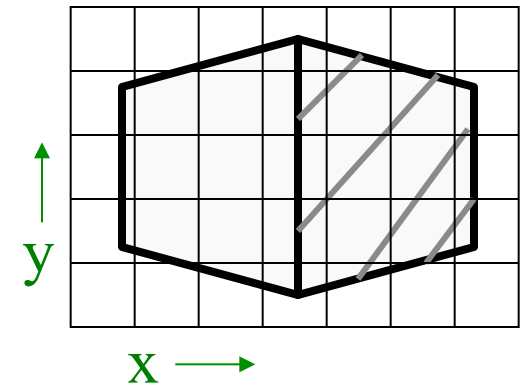
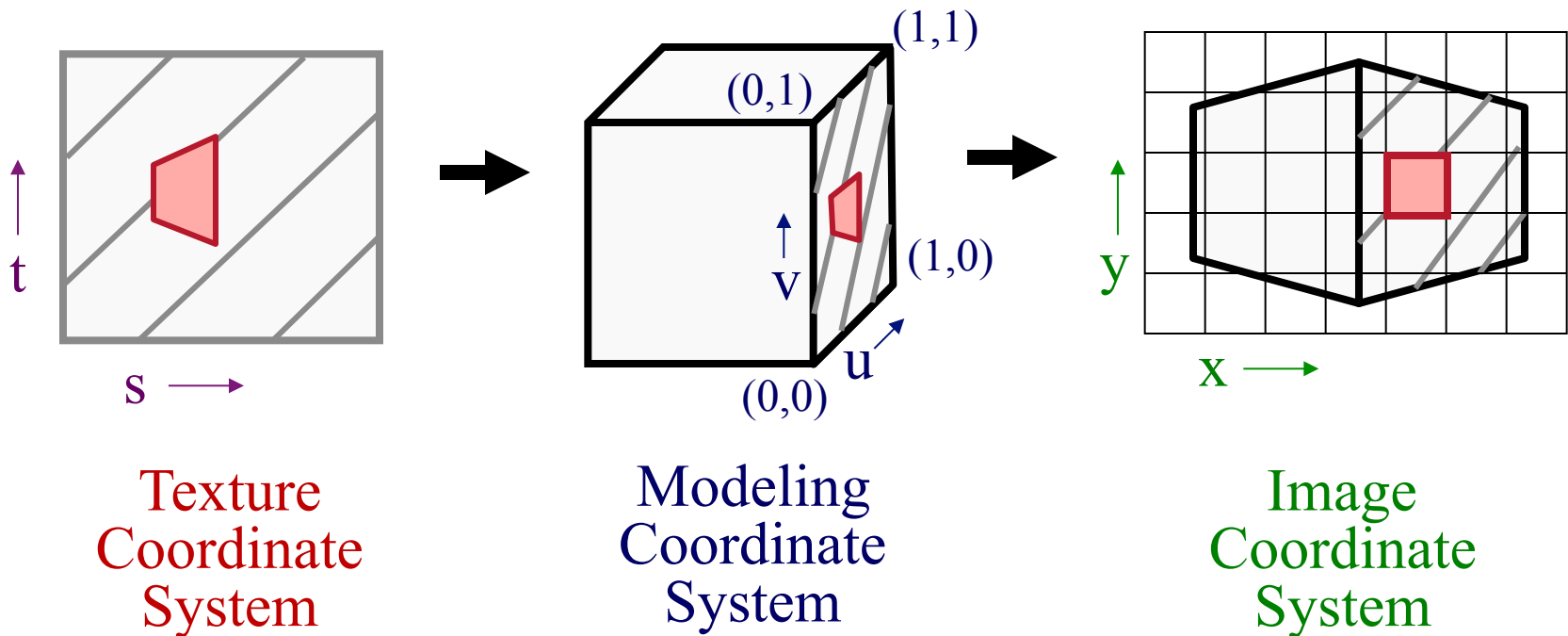


Image
Coordinate
System

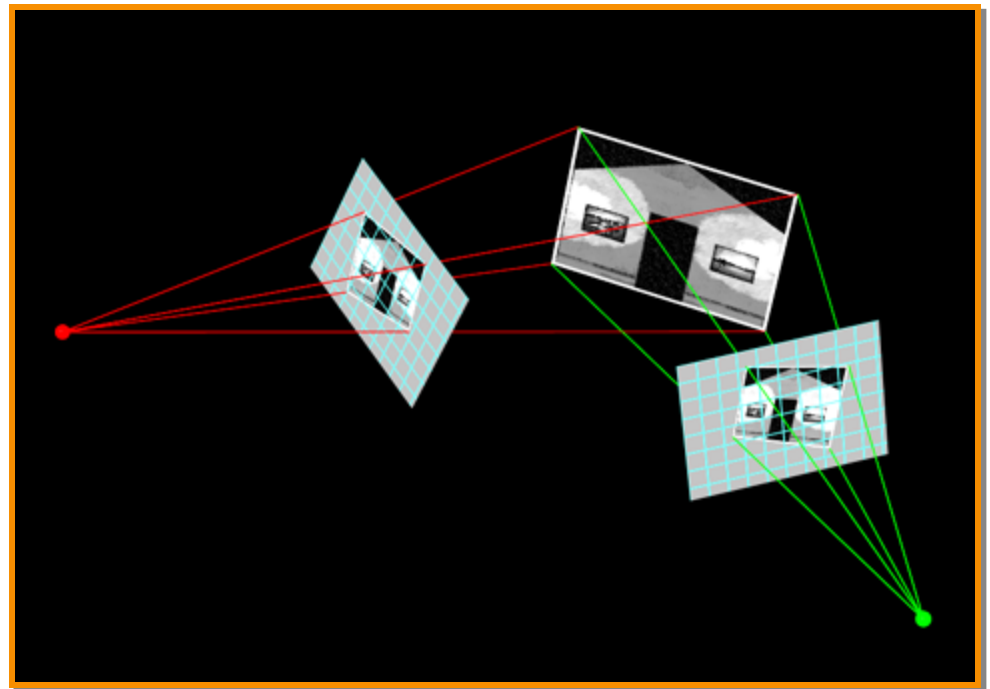
Texture Mapping

- When scan converting, map from ...
 - image coordinate system (x,y) to
 - modeling coordinate system (u,v) to
 - texture image (s,t)



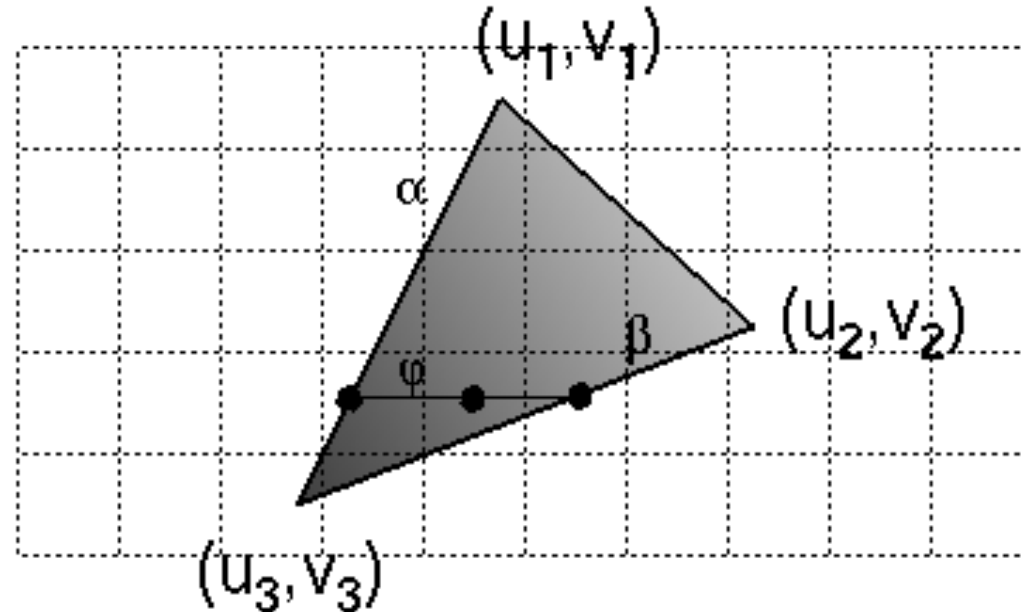
Texture Mapping

- Texture mapping is a 2D projective transformation
 - texture coordinate system: (s,t) to
 - image coordinate system (x,y)

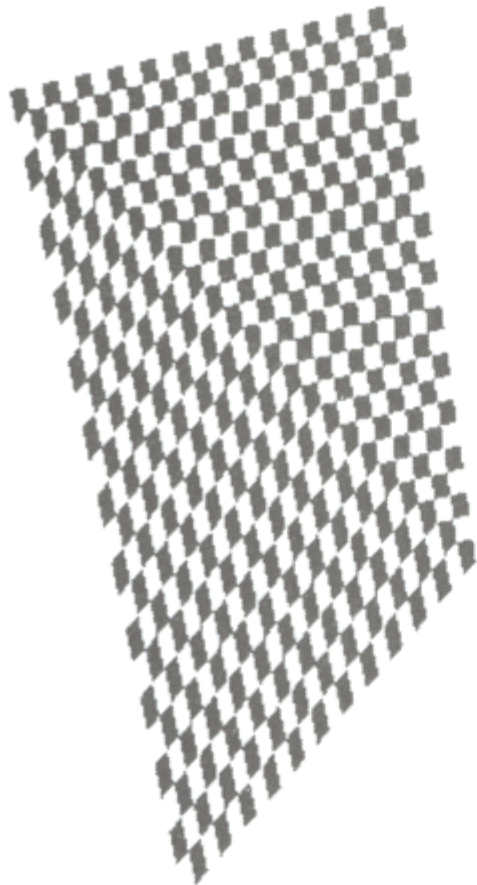


Texture Mapping

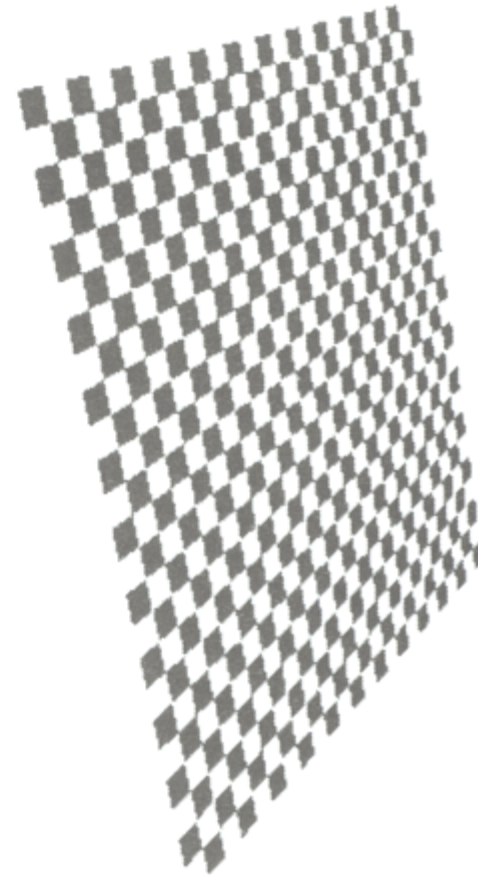
- Scan conversion
 - Interpolate texture coordinates down/across scan lines
 - Distortion due to bilinear interpolation approximation
 - » Cut polygons into smaller ones, or
 - » Perspective divide at each pixel



Texture Mapping



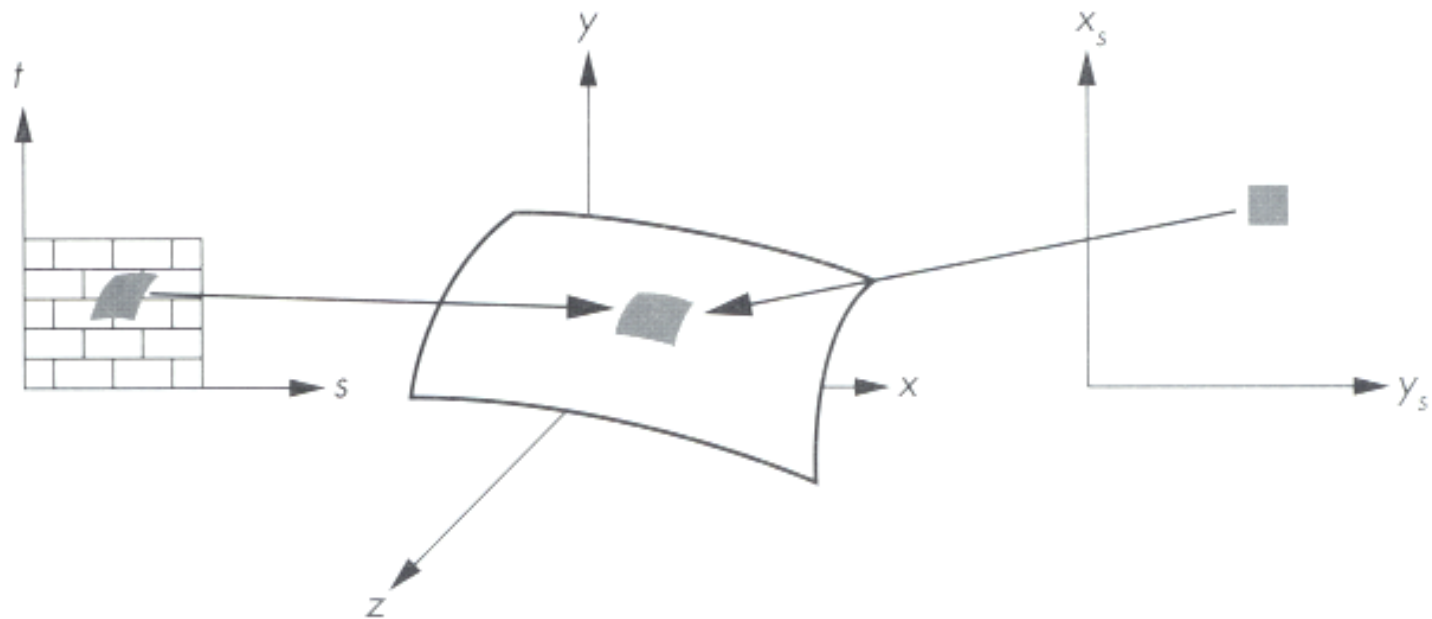
Linear interpolation
of texture coordinates



Correct interpolation
with perspective divide

Texture Filtering

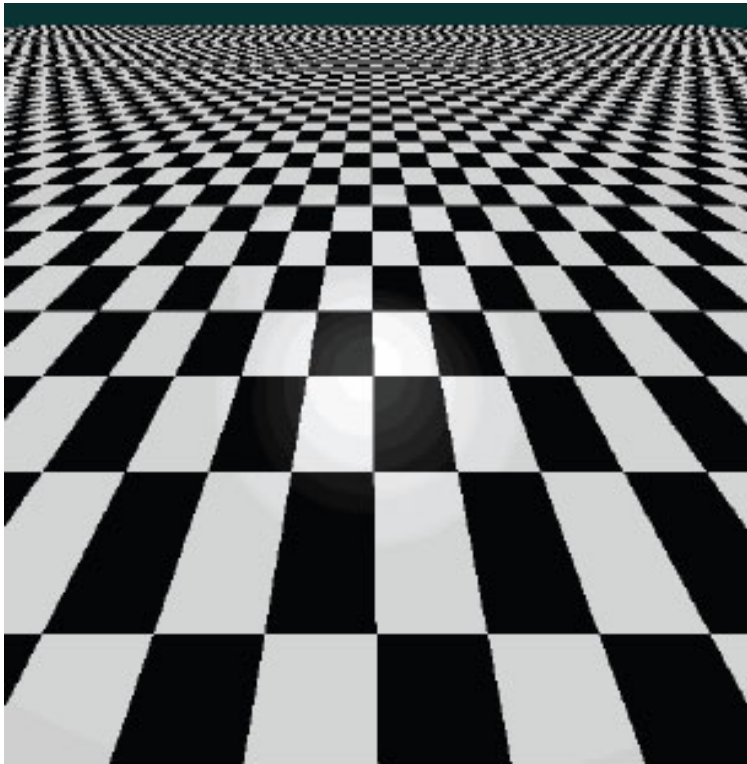
- Must **sample** texture to determine color at each pixel in image



Texture Filtering



- Aliasing is a problem



Point sampling

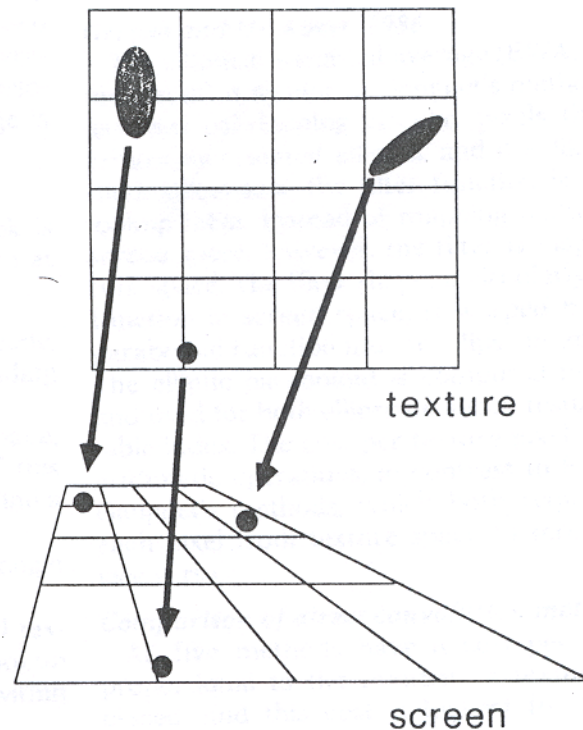


Area filtering

Texture Filtering



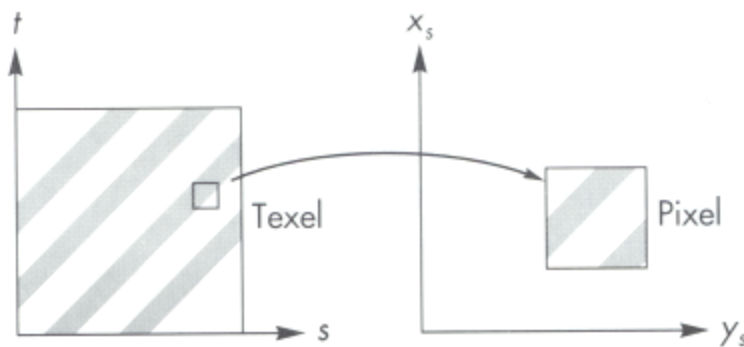
- Ideally, use elliptically shaped convolution filters



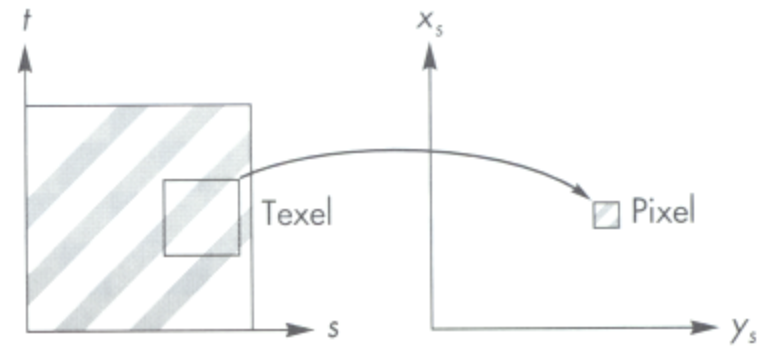
In practice, use rectangles or squares

Texture Filtering

- Size of filter depends on projective warp
 - Compute prefiltered images to avoid run-time cost
 - » Mipmaps
 - » Summed area tables



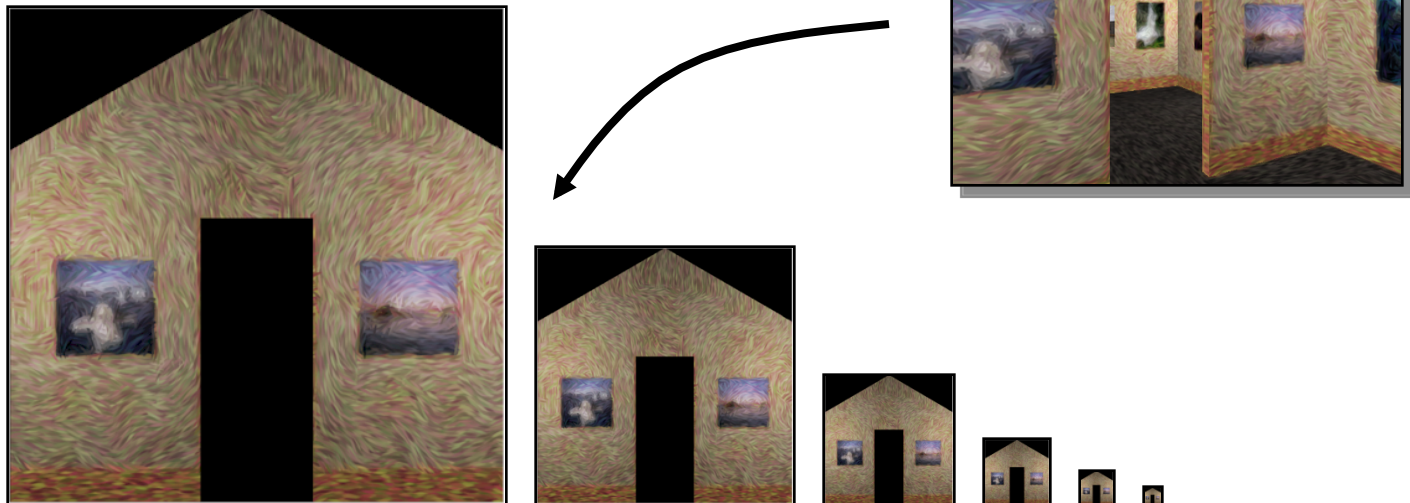
Magnification



Minification

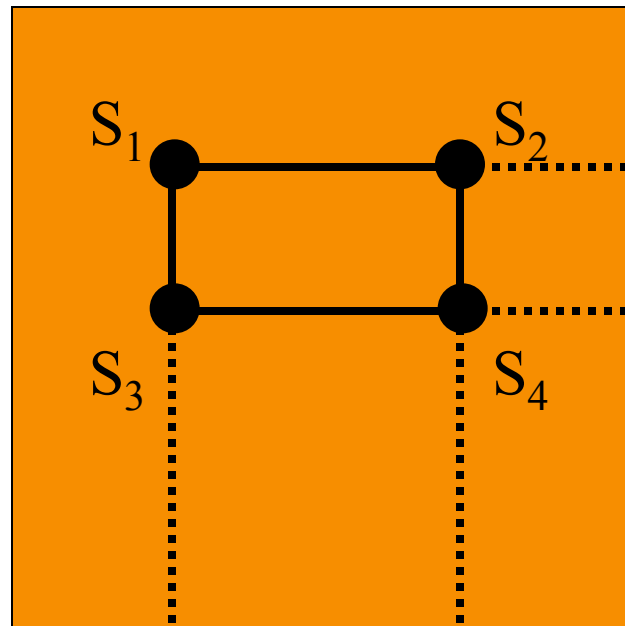
Mipmaps

- Keep textures prefiltered at multiple resolutions
 - Usually powers of 2
 - For each pixel, linearly interpolate between two closest levels (i.e., **trilinear** filtering)
 - Fast, easy for hardware



Summed-area tables

- At each texel keep sum of all values down & right
 - To compute sum of all values within a rectangle, simply combine four entries: $S_1 - S_2 - S_3 + S_4$
 - Better ability to capture oblique projections, but still not perfect

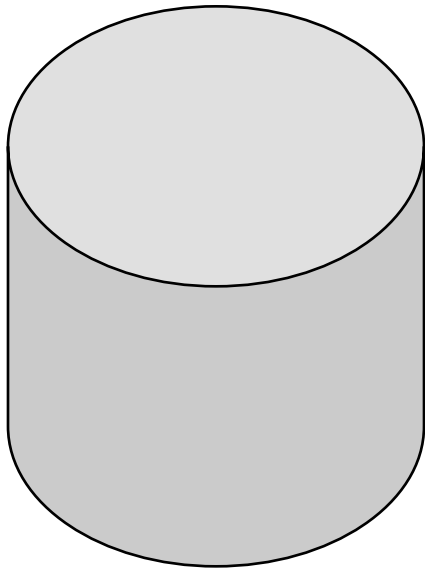




Texture Mapping Overview

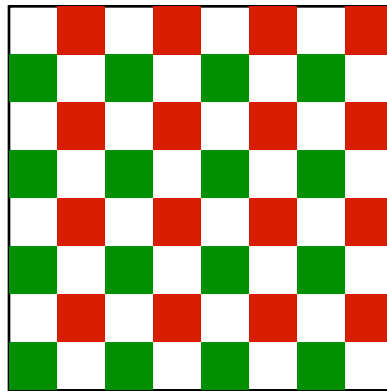
- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering

Parameterization



geometry

+



image

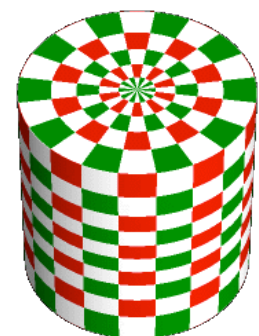
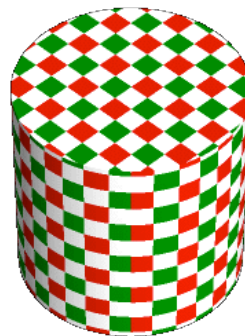
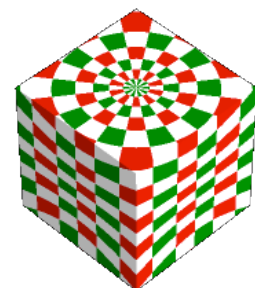
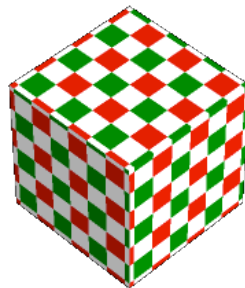
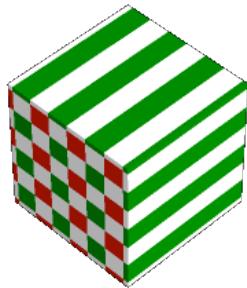
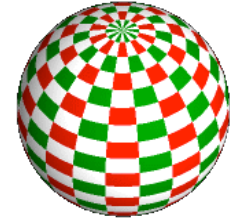
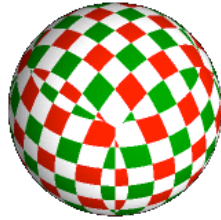
=



texture map

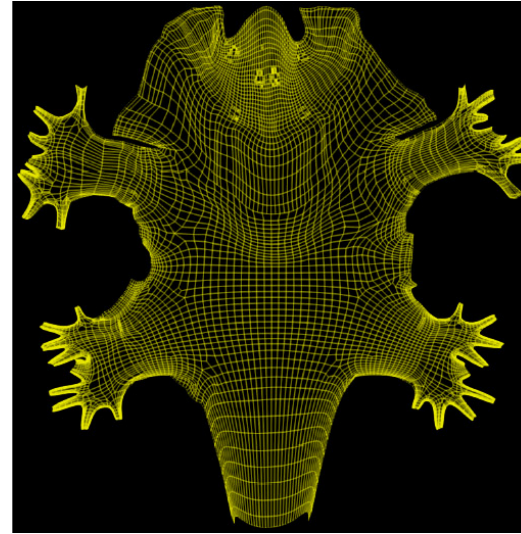
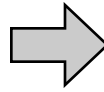
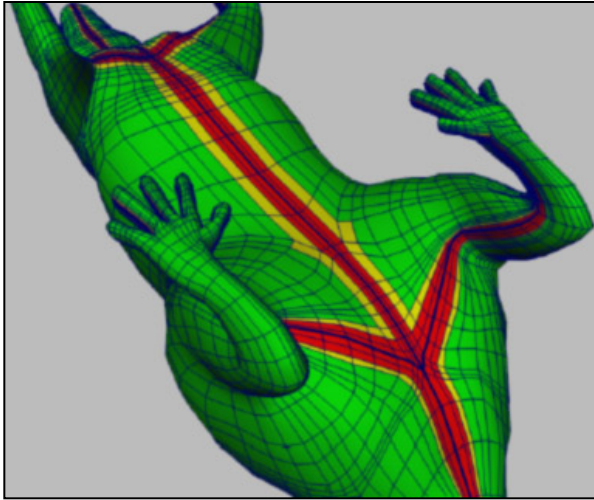
- Q: How do we decide *where* on the geometry each color from the image should go?

Option: function gives projection

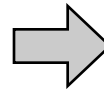
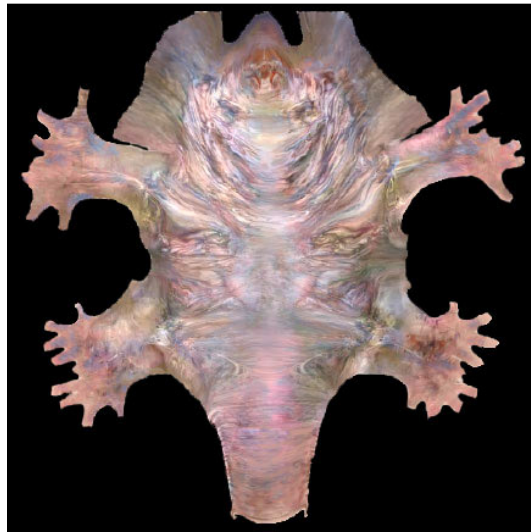


[Paul Bourke]

Option: unfold the surface



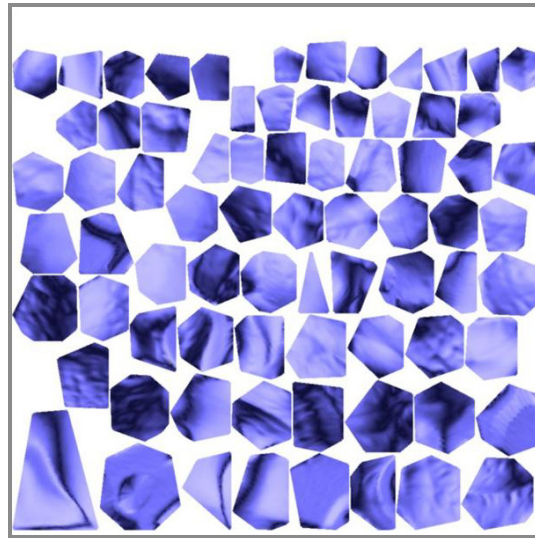
[Piponi2000]



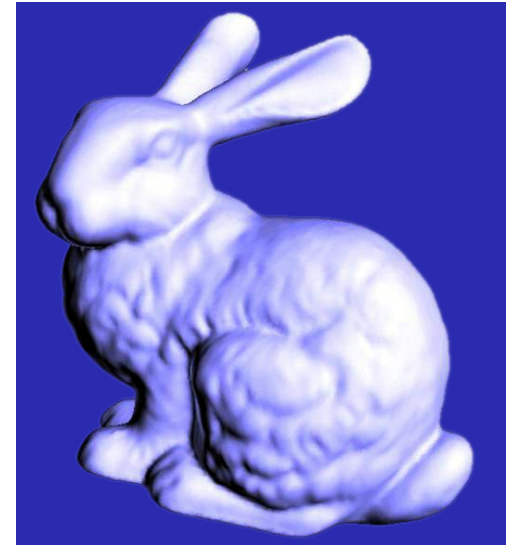
Option: make an atlas



charts



atlas



surface

[Sander2001]

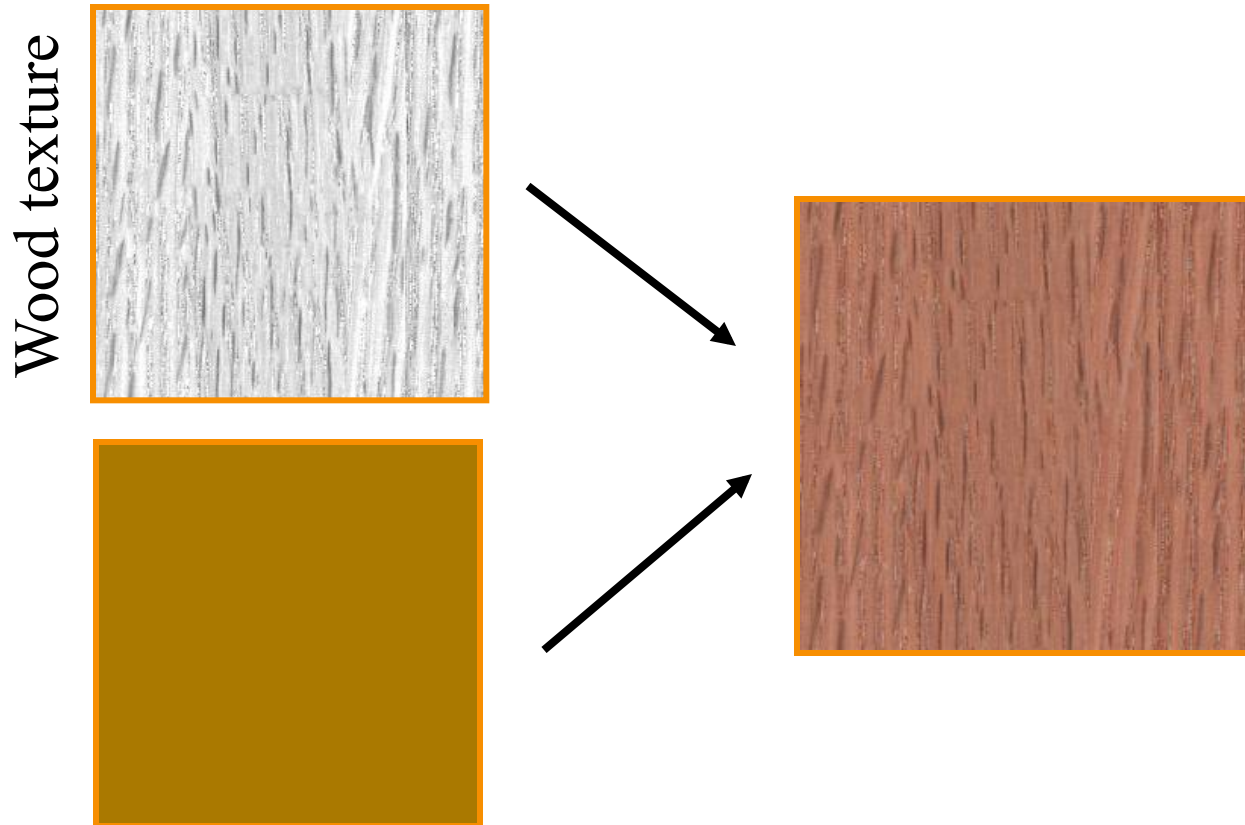


Texture Mapping Overview

- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering

Modulation textures

Texture values scale result of lighting calculation



$$I = T(s, t) \left(I_E + K_A I_A + \sum_L \left(K_D (N \cdot L) + K_S (V \cdot R)^n \right) S_L I_L + K_T I_T + K_S I_S \right)$$

Illumination Mapping

Map texture values to surface material parameter

- K_A
- K_D
- K_S
- K_T
- n

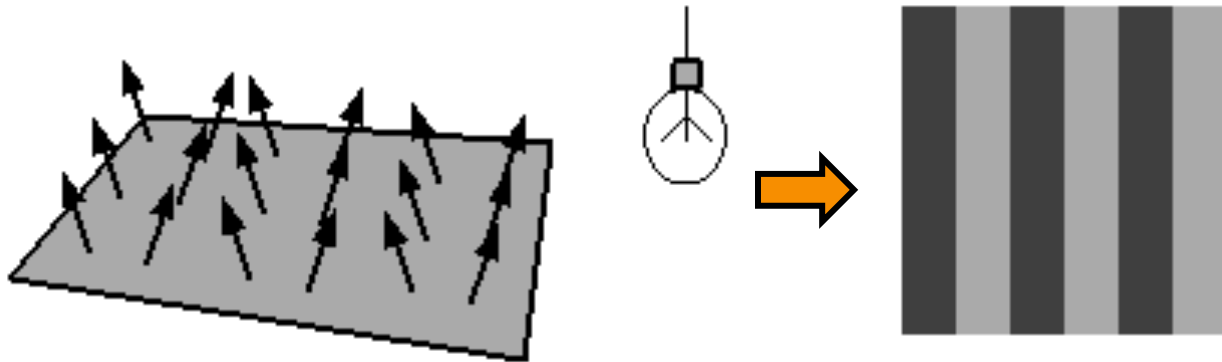


Texture
value

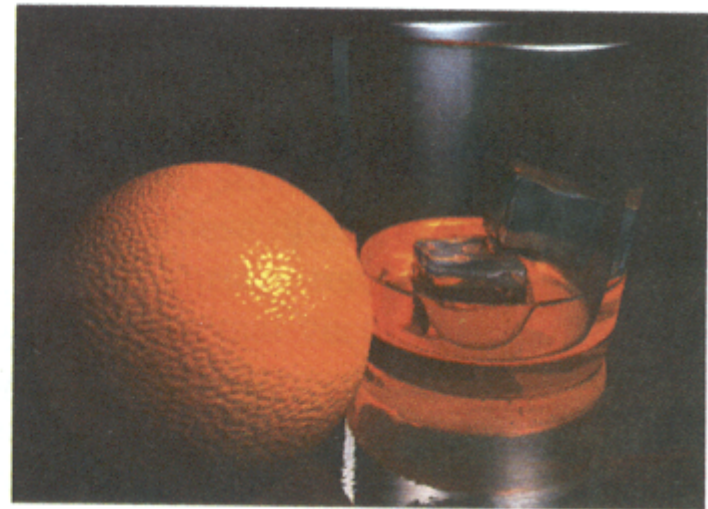
$$I = I_E + K_A I_A + \sum_L \left(K_D(s, t)(N \cdot L) + K_S(V \cdot R)^n \right) S_L I_L + K_T I_T + K_S I_S$$

Bump Mapping

Texture values perturb surface normals



Bump Mapping



Environment Mapping

Texture values are reflected off surface patch

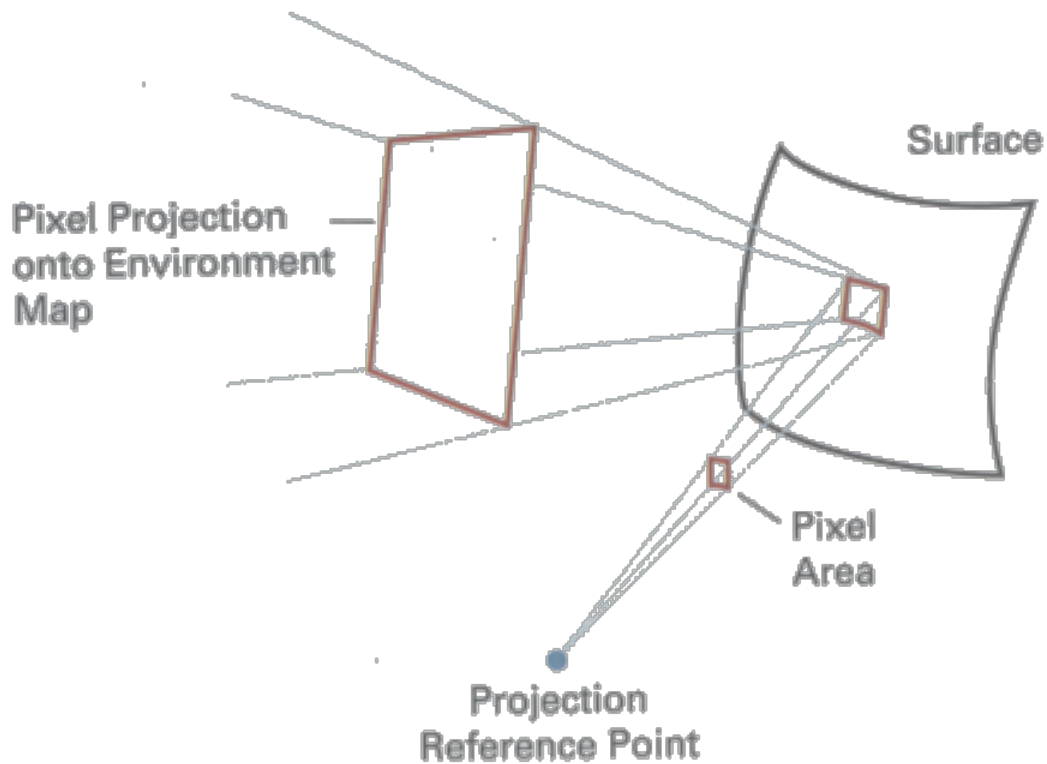
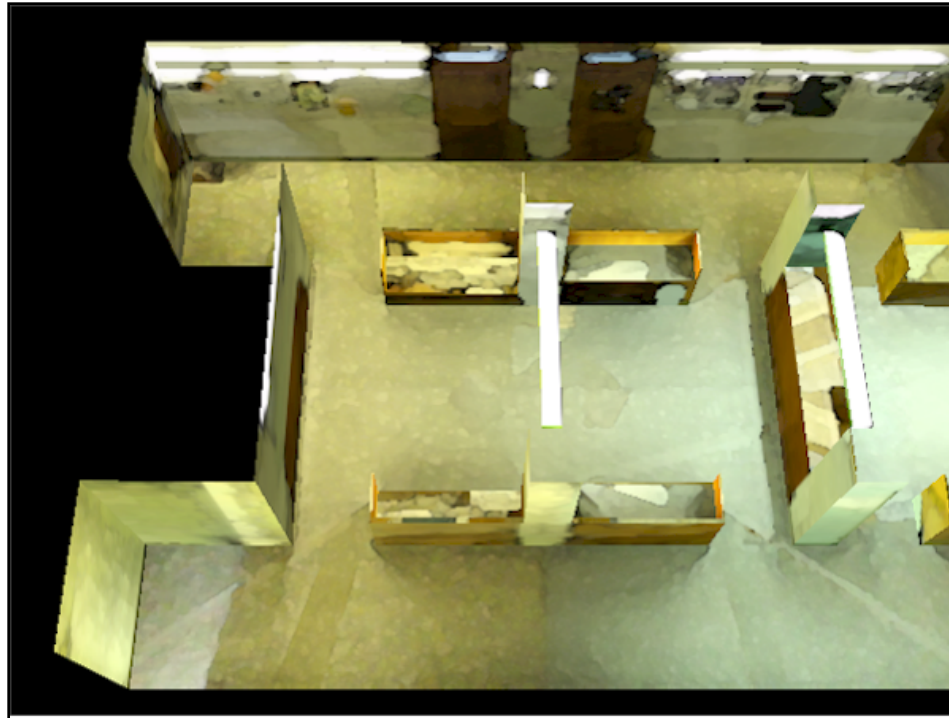


Image-Based Rendering



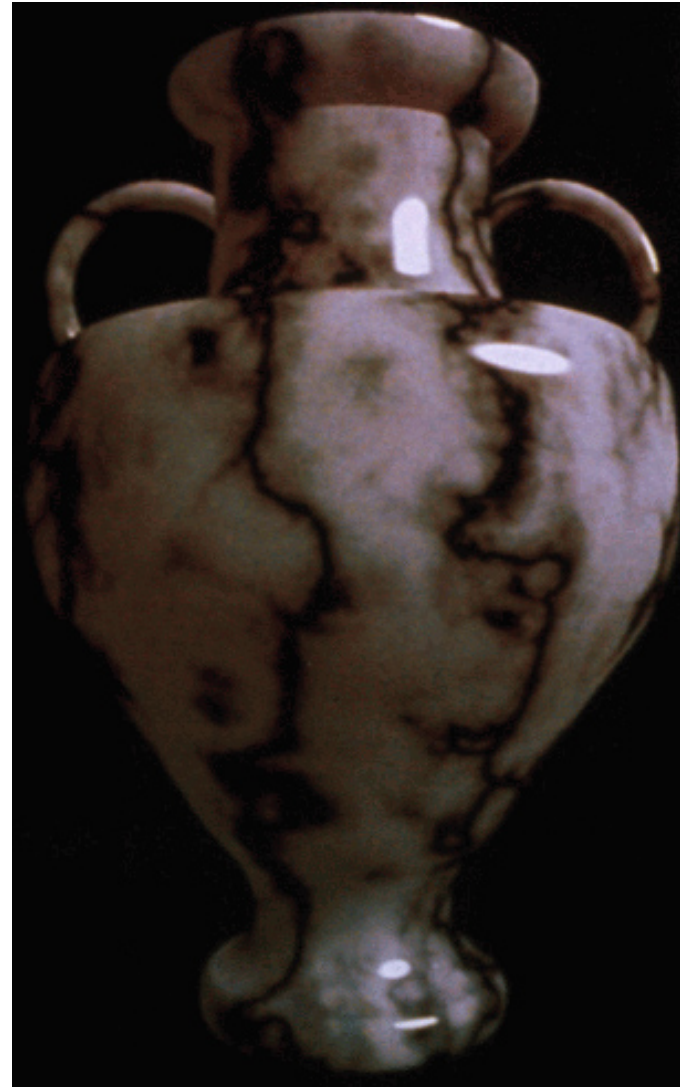
Map photographic textures to provide details for coarsely detailed polygonal model



Solid textures

Texture values indexed
by 3D location (x,y,z)

- Expensive storage, or
- Compute on the fly,
e.g. Perlin noise →





Texture Mapping Summary

- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Volume textures



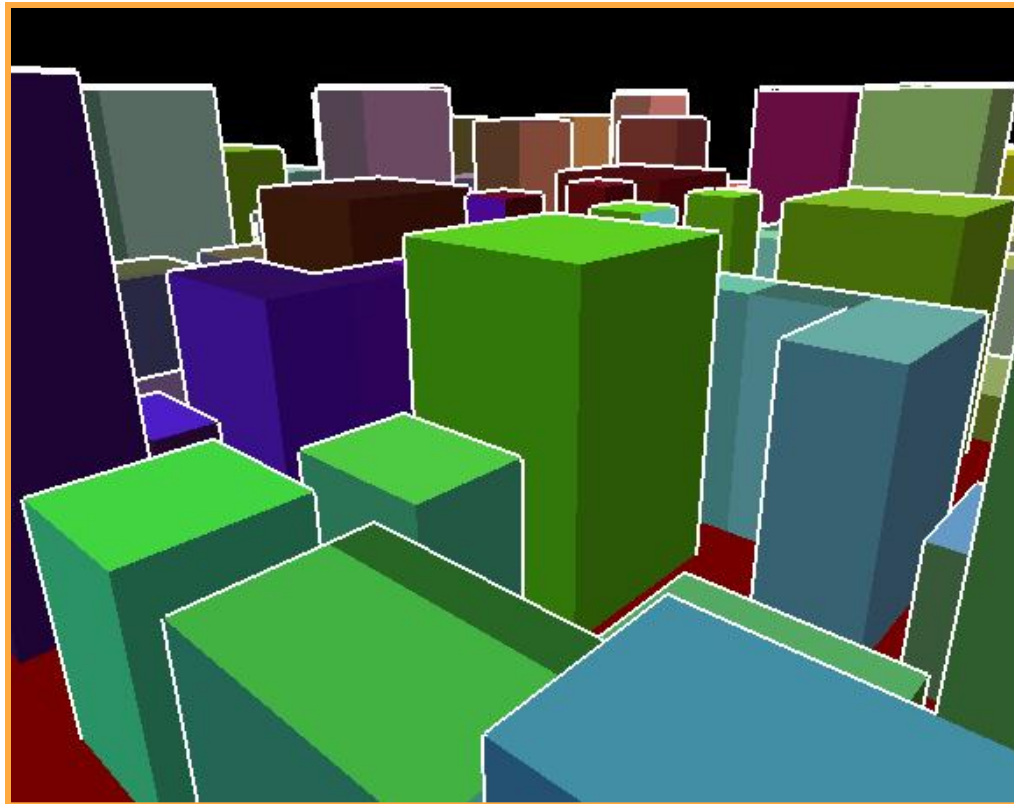
Rasterization

- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- **Visible surface determination**
 - Figure out which surface is front-most at every pixel



Visible Surface Determination

Make sure only front-most surface contributes to color at every pixel

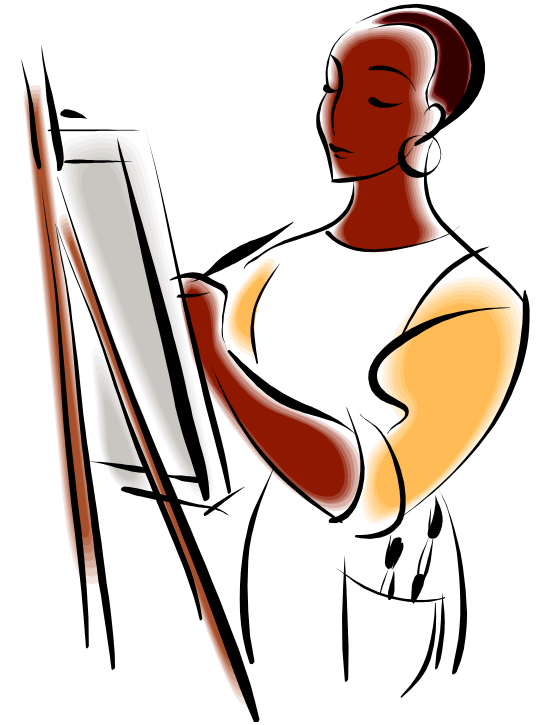
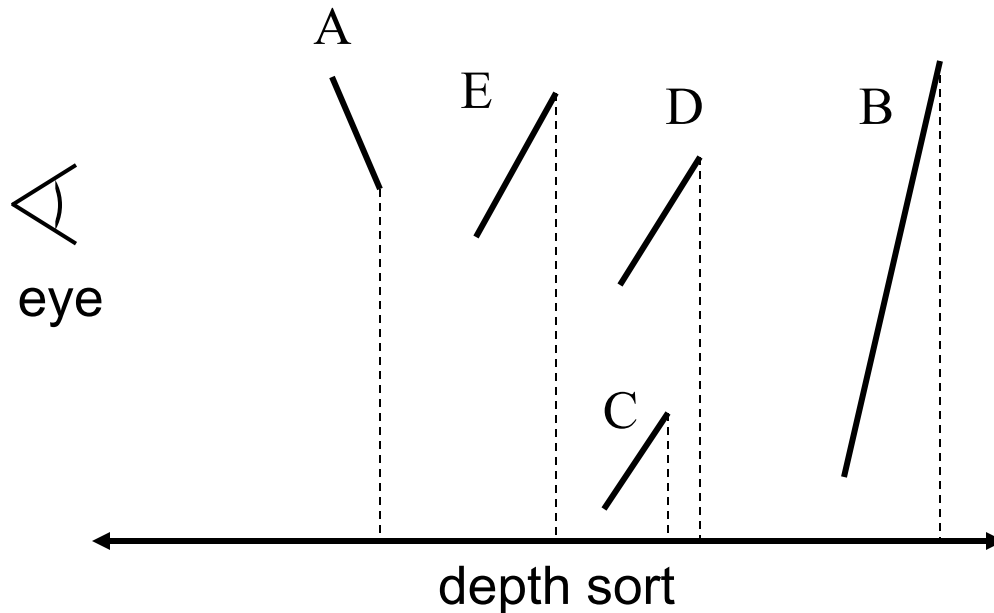


Depth sort



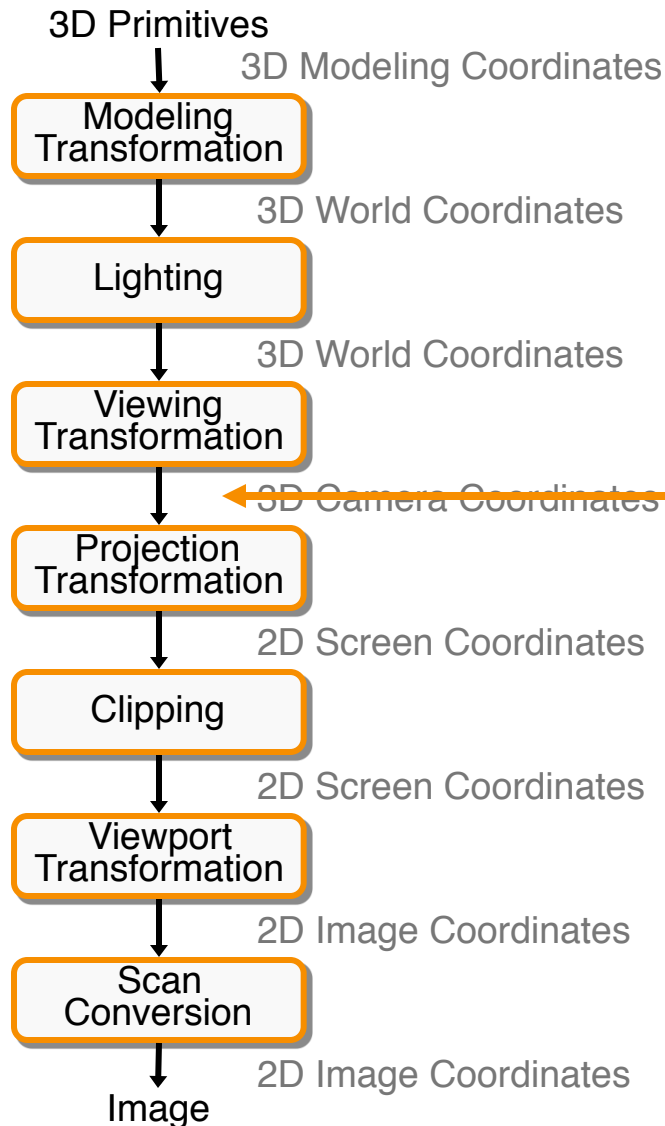
“Painter’s algorithm”

- Sort surfaces in order of decreasing maximum depth
- Scan convert surfaces in **back-to-front** order, **overwriting** pixels





3D Rendering Pipeline



Depth sort

Depth sort comments

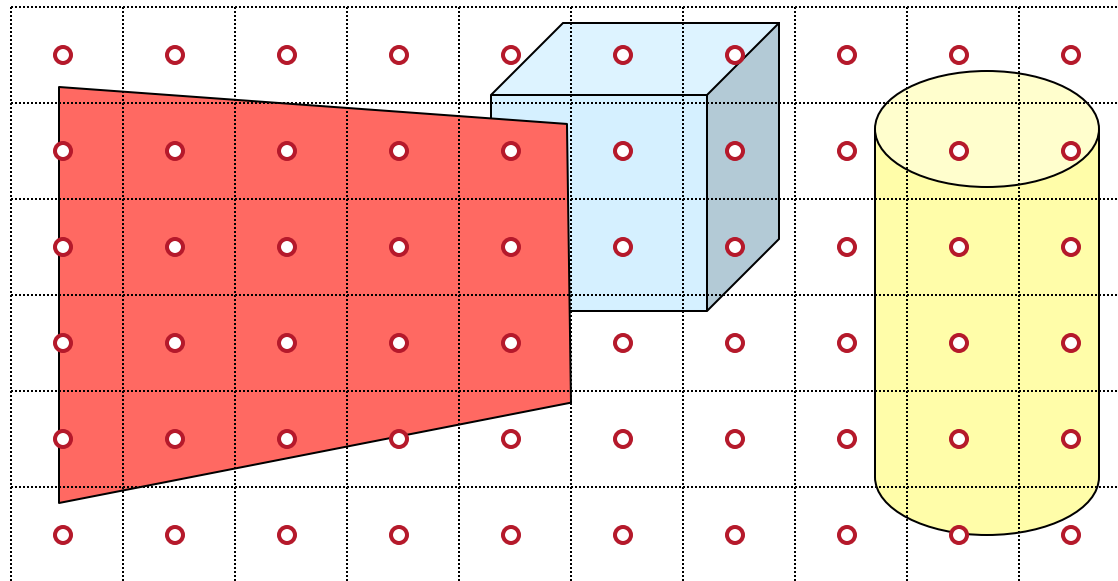
- $O(n \log n)$
- Better with frame coherence?
- Implemented in software
- Render every polygon
- Often use BSP-tree or static list ordering

Z-Buffer

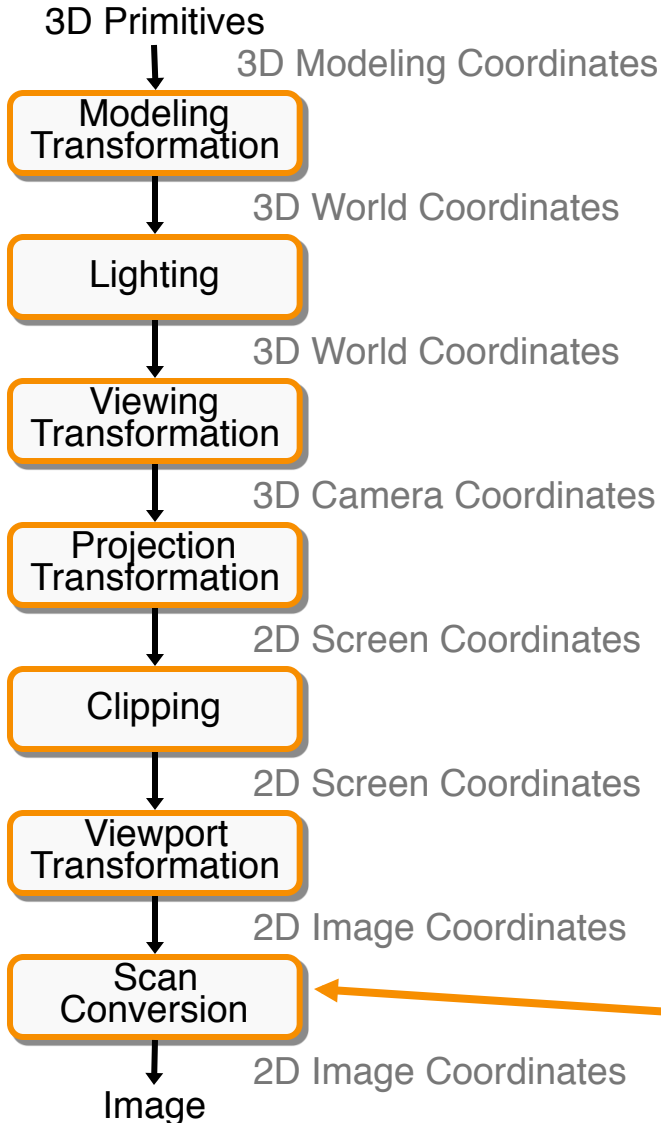


Maintain color & depth of closest object per pixel

- Framebuffer now RGBA_z – initialize z to far plane
- Update only pixels with depth closer than in z-buffer
- Depths are interpolated from vertices, just like colors

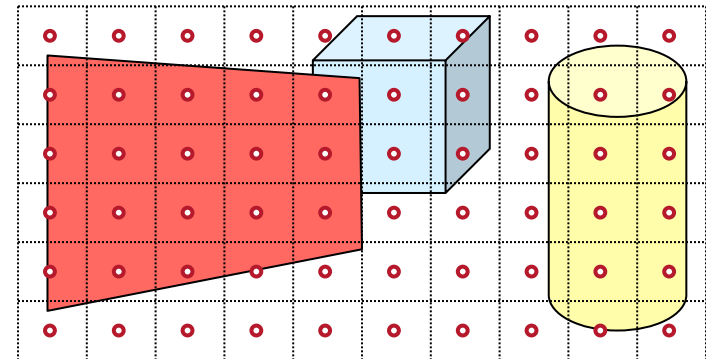


Z-Buffer



Z-buffer comments

- + Polygons rasterized in any order
- + Process one polygon at a time
- + Suitable for hardware pipeline
- Requires extra memory for z-buffer
- Subject to aliasing (A-buffer)
- o Commonly in hardware



Z-Buffer

Hidden Surface Removal Algorithms

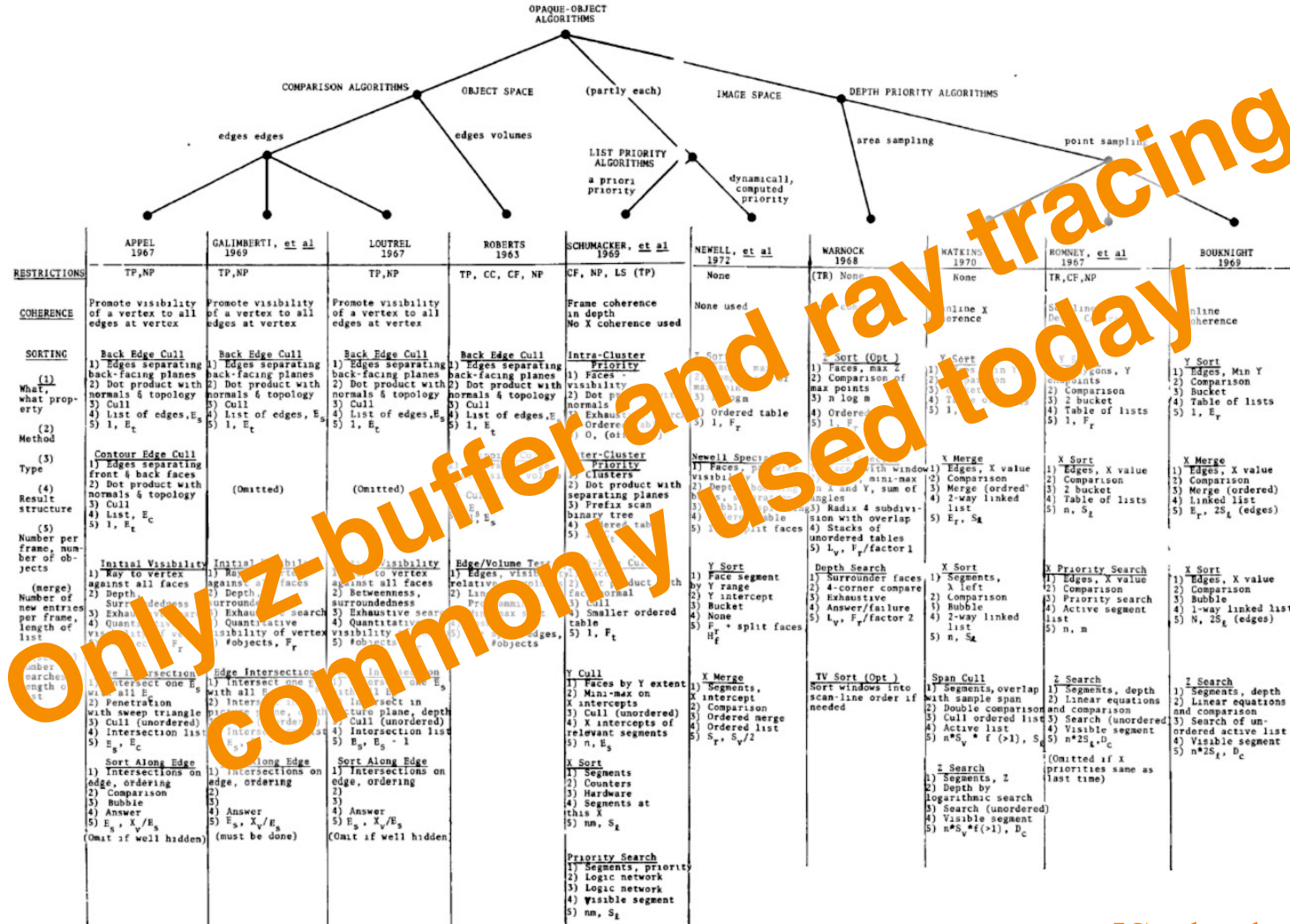


Figure 29. Characterization of ten opaque-object algorithms & Comparison of the algorithms.

[Sutherland '74]

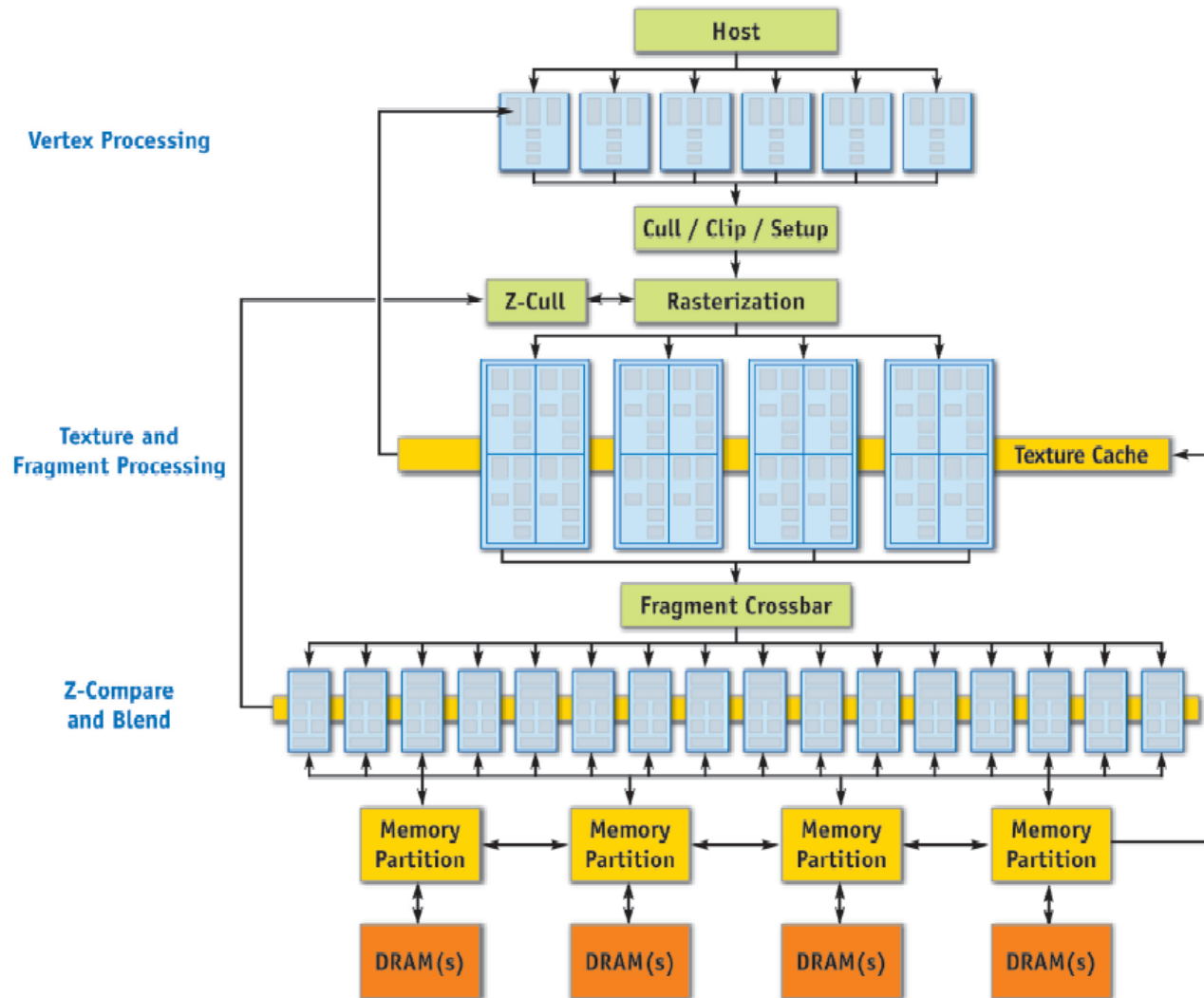


Rasterization Summary

- Scan conversion
 - Sweep-line algorithm
- Shading algorithms
 - Flat, Gouraud
- Texture mapping
 - Mipmaps
- Visibility determination
 - Z-buffer

This is all in hardware

GPU Architecture

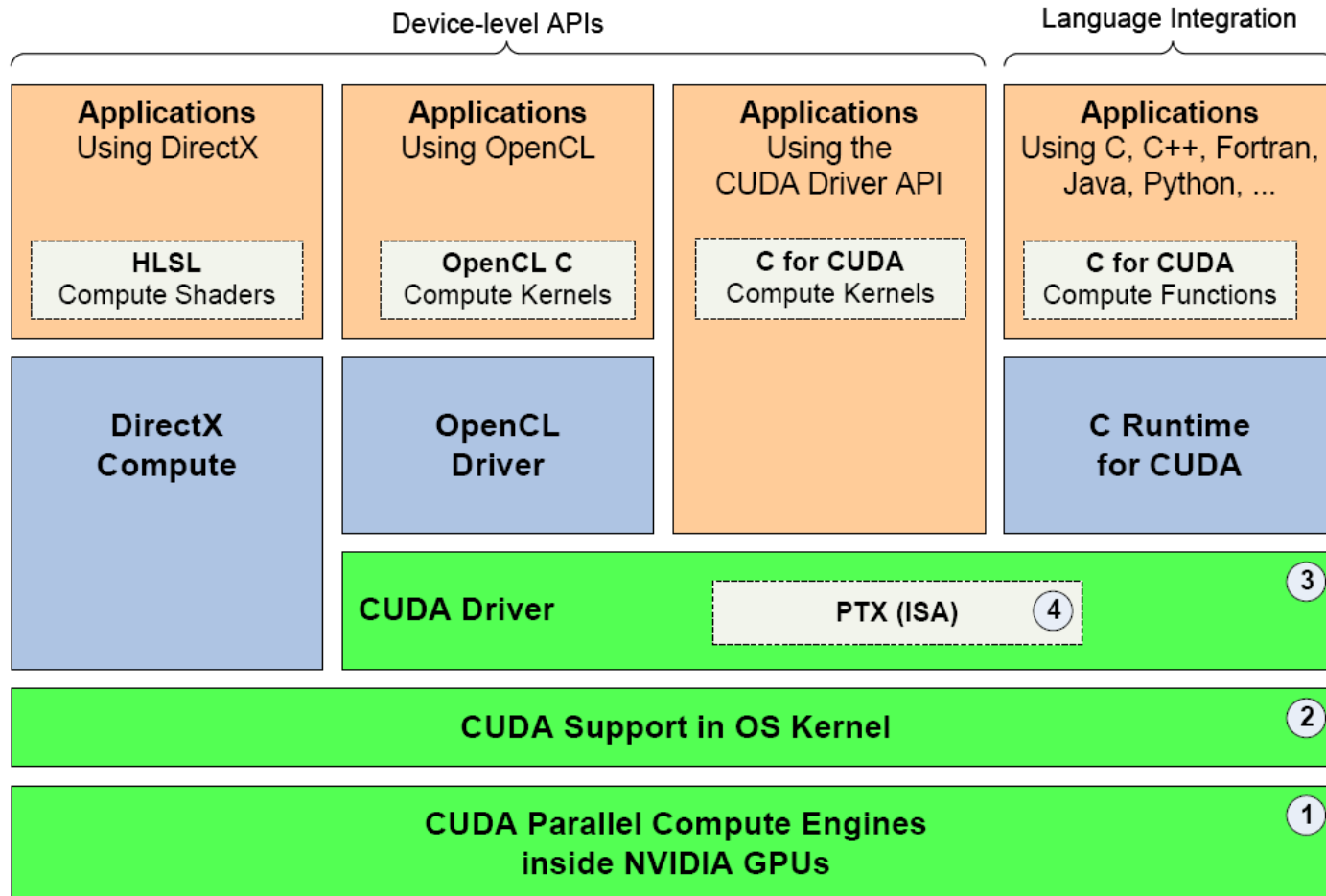


GeForce 6 Series Architecture

Actually ...



- Graphics hardware is programmable



Trend ...



- GPU is general-purpose parallel computer

